

①

AD-A280 600



NAVAL POSTGRADUATE SCHOOL MONTEREY, CALIFORNIA



THESIS

DTIC QUALITY INSPECTED 2

**Preliminary PANSAT Ground Station Software
Design and Use of an Expert
System to Analyze Telemetry**

by

Gregory Wade Lawrence

March 1994

Thesis Advisor:

I. Michael Ross

**DTIC
ELECTE
JUN 22 1994
S G D**

Approved for public release; distribution is unlimited

94-19087



13086

94 6 22 023

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY	2. REPORT DATE MARCH 1994	3. REPORT TYPE AND DATES COVERED Master's and Engineer's Thesis		
4. TITLE AND SUBTITLE PRELIMINARY PANSAT GROUND STATION SOFTWARE DESIGN AND USE OF AN EXPERT SYSTEM TO ANALYZE TELEMTRY		5. FUNDING NUMBERS		
6. AUTHOR(S) GREGORY WADE LAWRENCE				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER		
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		12b. DISTRIBUTION CODE A		
13. ABSTRACT (maximum 200 words) The Petite Amateur Navy Satellite (PANSAT) is a communications satellite designed to be used by civilian amateur radio operators. A master ground station is being built at the Naval Postgraduate School. This computer system performs satellite commands, displays telemetry, trouble-shoots problems, and passes messages. The system also controls an open loop tracking antenna. This paper concentrates on the telemetry display, decoding, and interpretation through Artificial Intelligence (A.I.). The telemetry is displayed in an easily interpretable format, so that any user can understand the current health of the satellite and be cued as to any problems and possible solutions. Only the master ground station has the ability to receive all telemetry and send commands to the spacecraft; civilian ham users do not have access to this information. The telemetry data is decommutated and analyzed before it is displayed to the user, so that the raw data will not have to be interpreted by ground users. The analysis will use CLIPS imbedded in the code, and derive its inputs from telemetry decommutation. The program is an expert system using a forward chaining set of rules based on the expected operation and parameters of the satellite. By building the rules during the construction and design of the satellite, the telemetry can be well understood and interpreted after the satellite is launched and the designers may no longer be available to provide input to the problem.				
14. SUBJECT TERMS PANSAT, Expert System, CLIPS, rule-based, ground station, telemetry, commanding, rule-based.			15. NUMBER OF PAGES 131	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

Approved for public release; distribution is unlimited

Preliminary PANSAT Ground Station Software Design
and Use of an Expert System to Analyze Telemetry

by

Gregory Wade Lawrence
Lieutenant, United States Navy
B.S., United States Naval Academy, 1986

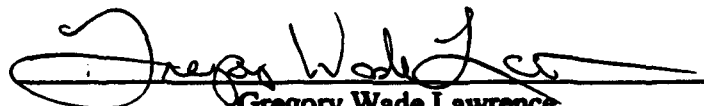
Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN ASTRONAUTICAL ENGINEERING
and
AERONAUTICAL AND ASTRONAUTICAL ENGINEER

from the

NAVAL POSTGRADUATE SCHOOL
March 1994

Author:


Gregory Wade Lawrence

Approved by:



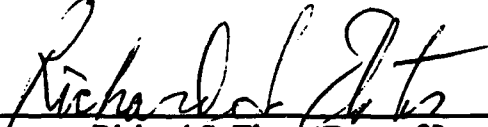
I. Michael Ross, Thesis Advisor



Neil C. Rowe, Second Reader



Daniel J. Collins, Chairman
Department of Aeronautics and Astronautics



Richard S. Elster, Dean of Instruction

ABSTRACT

The Petite Amateur Navy Satellite (PANSAT) is a digital communications satellite to be used by civilian amateur radio operators. The master ground station at the Naval Postgraduate School performs satellite commands, displays telemetry, trouble-shoots problems, passes messages, and controls an open loop tracking antenna. This paper concentrates on the telemetry subsystem, and interpretation with an Expert System. When commanding the satellite, the ground station software will verify the instruction with a ground-based simulator before it is sent to the satellite. Telemetry is displayed in an easily interpretable format, so that any user can understand the current health of the satellite and be cued to any problems and possible solutions. Only the master ground station has the ability to receive all telemetry and send commands to the spacecraft; civilian ham users do not have access to this information. The telemetry data is decommutated and analyzed before it is displayed to the user, so that the raw data will not have to be interpreted by ground users. The analysis will use C Language Integrated Production System (CLIPS) imbedded in the code, and derive its inputs from telemetry decommutation. The program is an expert system using a forward chaining set of rules based on the expected operation and parameters of the satellite. By building the rules early in construction and design satellite, the telemetry can be well understood and interpreted after the satellite is launched and the designers may no longer be available to provide input to the problem.

Accession For	
NTIS CRA&I	<input checked="checked" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
DTIC TAB	
Unannounced Justification	
By _____	
Distribution / _____	
Availability Codes	
Dist	Avail and/or Special
A-1	

TABLE OF CONTENTS

I. EXPERIENCE TOUR AND LITERATURE SEARCH.....	1
II. ARTIFICIAL INTELLIGENCE.....	7
III. CLIPS EXPERT SHELL	10
A. SEARCH STRATEGIES.....	11
B. DEPTH FIRST SEARCH.....	11
C. BREADTH FIRST SEARCH	13
IV. ORBITAL DYNAMICS	15
A. ORBITAL THEORY.....	15
B. NEWTON'S LAWS.....	16
C. TWO LINE KEPLER ELEMENTS.....	18
D. ORBITAL PERTURBATIONS.....	18
E. LOW EARTH ORBITS.....	20
F. DETERMINATION OF TIME IN VIEW.....	21
G. DETERMINATION OF TIME IN ECLIPSE	22
H. SIGNIFICANCE TO GROUND STATION AND A. I. ROUTINES.....	24
V. THERMAL CONTROL.....	25
A. LAWS OF THERMAL RADIATION.....	25
B. THERMAL CONTROL OF PANSAT.....	29
C. THERMAL DECISIONS	29
VI. POWER SYSTEM.....	31
A. SOLAR ARRAY DESIGN.....	31
B. DESIGN OF BATTERY	33
C. BATTERY CIRCUITS.....	35
VII. SATELLITE CONFIGURATION	40
VIII. CLIPS TELEMETRY ANALYSIS RULE SET	42

IX. GROUND STATION SOFTWARE DESIGN.....	47
A. REQUIREMENTS	47
B. GROUND STATION DESIGN.....	48
C. FILE TRANSFER	52
D. ADD PANSAT FILE HEADER	53
E. TELEMETRY SUBSYSTEM.....	54
X. SPACECRAFT COMMANDING	56
XI. CONCLUSIONS AND FURTHER STUDY	59
XII. LIST OF REFERENCES.....	60
APPENDIX A.....	62
APPENDIX B.....	93
APPENDIX C.....	126
INITIAL DISTRIBUTION LIST	131

I. EXPERIENCE TOUR AND LITERATURE SEARCH

There have been several satellite programs which have used Artificial Intelligence in the design of the system. Their use has varied as much as the satellites.

The first system studied was during a 6 week experience tour at the Navy Satellite Operations Center (NAVSOC), Pt. Mugu as part of the Naval Postgraduate School's curriculum. The center used to be the primary control center for the Transit navigation satellite constellation. Since the advent of GPS and the obsolescence of Transit, the center has converted to controlling several other satellites, including FleetSatCom, UFO (UHF Follow On), and Geosat. The center is designing a new ground station terminal which includes several of the features the PANSAT ground station has, including the Artificial Intelligence approach to analyzing the telemetry stream. Currently, due to shrinking funds, the A.I. analysis has stopped, but the rest of the system is becoming operational. The A.I. system was to be implemented with Fortran as a series of "if-then" statements. By using Fortran, NAVSOC had to write software that not only implemented the rules, but also stored the assertions, and develop the conflict resolution routines to determine which rule would fire. By using the C Language Integrated Production System (CLIPS), an Expert System shell developed by NASA, the programmer only needs to concern himself with the knowledge base. The inference processing is inherent in the software background.

Similar to the PANSAT ground station, the Integrated Satellite Control System (ISCS) at NAVSOC is menu driven with real time graphics for trending analysis, and hierarchy display models. The system controls several satellites, so it goes one step higher than the PANSAT ground station in the hierarchical display of information. It combines all telemetry points per satellite into one status when displayed. If the user wants amplifying information on a particular satellite, he selects it, and can view more detailed information about each subsystem and telemetry point.(Ref. 1) ISCS differs from the

PANSAT system in that it performs analysis only when a limit has been exceeded. Currently, this analysis is performed by the Satellite Managers, without the aid of software.

An A.I. system was developed for Canada's Radarsat which could bear many similarities to PANSAT's use of computer reasoning. The satellite is a civilian low earth orbiting radar platform, so communications with the satellite are infrequent and of short duration. The system must operate with failed or degraded components, and short term power demands while the radar is transmitting exceed the power output of the solar panels. Depending upon the tasking of the satellite, it may be commanded more than it can possibly achieve. Each radar pulse will drain the batteries, and if too many missions are tasked, the electrical power system will need to determine the energy available. The power system is greatly influenced by the combination of the duration and frequency of eclipse, and the duty cycle of the payload. Since the power budget is well known on the ground, the tasking software uses rule based decisions to command the satellite without exceeding the power budget. Any faults not yet detected and accounted for by the ground station which affect the power system must then be autonomously handled by the spacecraft until the ground station can analyze the situation and make corrective adjustments in the satellite tasking procedures. The satellite must be able to operate autonomously up to 12 hours on a normal routine, and up to two weeks as a survival criterion.(Ref. 2)

A satellite workstation designed by the Aerospace Corporation for use in a military satellite testing operations is evaluating the use of expert systems. The company decided that their preliminary rule based system was too limited. The project was then modified to capture as much of the engineering and testing experience as possible, in a way that could be updated. The expandable library would then allow the users a much more thorough analysis with an ever changing expert system. With the large expected increase in satellites over the next decade, the Advanced Satellite Workstation (ASW) was designed to reduce the cost of ground control through reduced manpower requirements. An

integrated support system would require fewer specialists, and the repetitive processes would be handled by the workstation so the engineers could do more analysis. The system also performed computationally intensive processes, i.e. telemetry decommutation and display, with an off-line computer.(Ref. 3)

The ASW supplies several features to aid satellite operators, some of which could be added to the PANSAT ground station as the design evolves. One feature is the ability to quickly access satellite logic diagrams, simulations, and pictures of the satellite with hypermedia presentations to aid in visualization of the flight spacecraft. The telemetry is displayed very similar to the PANSAT in a windows format. All telemetry is color coded, and windows of telemetry data are easily viewed and manipulated. All commands are also screen buttons and menus for quick access. During a pass, both real time and past data are downloaded and analyzed with the rule based expert system. Currently the system uses around 100 rules which determine maximum and minimum limits, rate of change, and combinations of rules based on the satellite design and operational experience.(Ref. 3)

The Martin Marietta Corporation designed its own system for programming and maintaining expert systems, the Multi Purpose Causal (MPC) Tool. This system is based on Lisp, and added several features that make it very easy to use. The expert shell is set up in a windows environment so interaction with the system is very user friendly. It also has tools for numerical simulation of an operating system, and fault isolation software.(Ref. 4)

Several expert systems have also been developed for space applications through NASA. The systems are designed to reduce the burdensome tasking of the flight controllers, who constantly monitor the Space Shuttle telemetry. The increase in automated systems is also desired in anticipation of Space Station requirements, where the telemetry tasking will be significantly increased. Before the advent of intelligent software, the flight controllers scanned the binary and numerical data displayed in monochromatic format on their computer screen. They looked for familiar normal patterns and system behavior, pre-defined anomalous behavior, and unknown data sets. This was performed

through memorization and recall of explicit patterns, so any increase in workload drastically reduced the flight controller's ability to properly interpret the data with accuracy and efficiency. The flight controllers also suffer from vision fatigue, mental fatigue, and boredom. When they find an anomalous data reading, they must consult volumes of system information to isolate the fault and find steps to alter the mission according to the degraded system. The automation of this process frees the controllers from the repetitive task of pattern searching and telemetry interpretation, and can offer solutions in real-time to continue the mission after a system degradation is found. All decisions are based upon data from the telemetry stream. The syntax of the CLIPS programming was set in an orderly fashion to prevent or reduce ambiguous, and redundant rules. All facts are represented by fields describing the Object-Attribute-Variable (O-A-V). By presenting all facts in the same format, the rules could be more easily integrated and written. The types of facts were state, timing, telemetry. The assertions were categorized into state, status, output.(Ref. 5) These basic formats are very similar to those developed in the program for PANSAT.

The ultimate space application of an expert system is the Space Station. The number of systems, and amount of data will greatly outnumber any system now in use. The use of integrated, intelligent software will greatly reduce the number of highly trained personnel, and increase the productivity of the current set of flight controllers.(Ref. 6)

The Aerospace Corporation also designed a rule based expert system for determining false events in the telemetry data from an earth observing sensor. This program used information about the health of the satellite, the sensors, data processing, knowledge of the problem, and physical phenomena that affect the sensors. The company wanted to determine if A.I. techniques were feasible for satellite data considering the large amount, the variety, and the complicated nature of the system. The system has to distinguish between operational parameters and natural phenomena, such as reflections off ice-capped mountains, and its differences from other noise and signal sources. The expert system developed was successful in maintaining system knowledge, making quick

decisions, and giving explanations for those decisions, which increased the user's confidence in the expert system. The first stage was to provide an "intelligent assistant" to new and inexperienced display operators. This was accomplished by providing information normally only available from system engineers and analysts. The final goal was to be able to distinguish actual signals from the satellite from noise errors caused by reflections off ice capped mountains, sea surface scattering, and other phenomena. Like PANSAT, the expert system is data driven. In their system, the data was imagery from the satellite, while the PANSAT system uses telemetry data points. The Aerospace Corporation found that the expert system was a viable approach to false event determination. It surpassed the success of traditional signal processing techniques by an order of magnitude.(Ref. 7)

Another project that developed an expert system was the Oceanographic Expert System which could potentially be used on Tactical Environmental Support System, Third Generation, TESS(3) stations. The program was developed to predict mesoscale feature movements in the Gulf Stream as an aid for weather prediction from satellite imagery. The use of an expert system was determined based upon the observation that human experts can solve problems that conventional computer techniques cannot. The rule based system derives its inputs from kinematic aspects of the ocean surface, and predicts the "state" at a later time. While the system uses a rule based program for prediction of the future of the changes, it relies upon a transcendental equation for the location of the Gulf Stream to make its calculations. The coefficients of the equation are determined by a neural network, another type of Artificial Intelligence, which in this case reproduces a high order equation. The system was coded in a combination of OPS83, C, and Fortran. The programming structure most resembles PROLOG of the standard AI. languages. Work is underway to convert the code to CLIPS. In order to run the software on different computer systems, it must be converted to a standard language. The conversion from OPS83 to PROLOG would require significant changes, while CLIPS provides an

alternative with other advantages, since it can be embedded in procedural code and called as a subroutine, and is available to all government agencies and their contractors. (Ref. 8)

II. ARTIFICIAL INTELLIGENCE

The basic purpose of Artificial Intelligence is to represent knowledge in computer memory, capture the essential features of a problem, and make that information accessible to a problem solving algorithm. The system formally puts facts, rules, advice, and procedures into a computer where that can be manipulated to reach a final outcome. Knowledge representation is one of the major successes of artificial intelligence. The solutions to many problems depends more upon the availability of knowledge than sophisticated algorithms for determining solutions. The knowledge base or rules are computational objects and relationships that represent occurrences in the real world.(Ref. 9)

The telemetry analysis will use CLIPS imbedded in the C code, and derive its inputs from telemetry decommutation, and files containing system configuration and satellite derived decisions. The program is a forward chaining set of rules based on the expected operation and parameters of the satellite. By building the rules during the construction and design of the satellite, the telemetry will be very well understood after the satellite is launched and the designers are no longer available to provide input to the problem. The code will also be well documented so that rules can be changed, added, or deleted easily. The rules may not have the actual operating conditions well known since the spacecraft has not yet been built and tested, so the operating limits may change. Another likelihood is that a situation arises that was not expected in the design of the satellite, so a new rule will have to be written to account for this situation.

A expert system approach was used for several reasons. Knowledge can be added and modified easily without major reprogramming, and it has interactive capabilities for explanation, verification, and debugging, as well as independence of the knowledge base from the inference system. A rule based system can also reach many conclusions depending upon the data and the system. A satellite may have problems in any number of systems, and the analysis must find all conclusions. The set of rules is where the human

expert's knowledge about the operation of the satellite and its reflection in the telemetry data are retained. Rules can be easily constructed since most engineers and experts express problems in a method of "for a given situation, these actions must be performed". Each rule then represents an independent piece of knowledge, and new knowledge can be added in a modular fashion. The results of rules are assertions, the conclusions of a rule which add to the facts known, based upon the data. These assertions can also be called dynamic knowledge, since they are dependent upon the satellite telemetry, and are constantly changing. (Ref. 10)

The expert system also has a weakness. These knowledge representations are not robust. They cannot reach conclusions, or make best estimates of conclusions that were not known before hand. The knowledge base only represents the expert's mind and his experience. (Ref. 10) If a new situation arises which the expert has not encountered, he can't possible express a rule to search for the conclusion. Even if the expert can quickly reach a conclusion, the analysis system won't be able to until the rule base is updated. This deficiency makes some A.I. experts consider the term Expert System a misnomer, since the rule-based system behaves more like a novice that can only perform as it is commanded.(Ref. 11)

The operation of most A.I. software can be broken down into two different categories, backward chaining and forward chaining, which depends upon the type of logical procedures used. Prolog is an example of a backward chaining inference engine, while CLIPS uses forward chaining inference.

Backward chaining is defined as the procedure to reason from goals to facts, reducing goals to subgoals along the way, and hoping that all subgoals are reduced to facts eventually. This type of processing is also known as goal-driven search, top-down inference, and consequent reasoning. The steps the computational engine uses to solve backward chaining systems is difficult to follow. Backward chaining should be used when the conclusion is given, or can easily be formulated, there is a large number of rules that

match facts, want only one conclusion, or problem data are not available and must be acquired by the problem solver.(Ref. 9)

Forward chaining is defined as the process of reasoning from facts to goals, producing new facts along the way, and hoping the goals can be generated. This type of process is also known as data driven search, bottom up search, and antecedent reasoning.(Ref. 9)

The procedure for forward chaining is simple to understand. The processing engine scans the database for the set of applicable rules. As long as there are rules, and the set is not empty, then the final goal has not been reached. Then the computer selects from the set a rule whose conditions are satisfied. It applies the rule, and updates the database if necessary. Then the software scans the data base for applicable rules. It selects another rule, and continues the process until there are no rules whose conditions are satisfied. At this time, the search has probed all possible paths.

III. CLIPS EXPERT SHELL

CLIPS can be thought of as an "expert-system shell" or "knowledge engineering tool" rather than a programming language. The system provides the tools needed to build a rule based system while keeping the inference processing part of the software hidden from the user. CLIPS was developed by NASA to be easily embedded in C, and to use syntax very similar to the standard set by Lisp.

A programmer would want to use forward chaining over other types of software when his problem has certain characteristics. A forward chaining engine should be used when all of the facts are given, there is a large number of potential conclusions, the user wants to reach every possible conclusion, and it is difficult to form a conclusion. All of these characteristics of a forward chaining, rule based system are the desired for the analysis of telemetry. NASA has developed the CLIPS software after many years of experience with satellites, and numerous other real-time operational projects. CLIPS is also designed so that it can be modified very easily, without affecting the rest of the system, and the engineer can concentrate on the capturing of knowledge rather than the low level implementation of a rule system.(Ref. 9) The design constraints of PANSAT require that a personal computer be used for the ground station and written in C and CLIPS is optimized for a smaller machine. Taking all of these factors into account, the software for building an Expert System to analyze PANSAT telemetry is CLIPS.

A CLIPS program consists of three major components. The first is the knowledge base. This base is the source of system knowledge including all known facts and rules that determine relationships between those facts. The next important aspect is the inference engine. This component is the interpreter for the knowledge base, and applies the knowledge to the solution of actual problems. The last component is the user interface, which makes access more comfortable and hides all of the system complexities. There are

many variations in the user interface, including menus, question and answer, language, commands, and graphical interfaces.(Ref. 9)

A. SEARCH STRATEGIES

The software must search through the rule sets to reach a final conclusion. This search is a systematic routine that explores the problem space in search of a goal state. There are several different search strategies that can be used. The two most important are the depth first and breadth first strategy. In the simplest sense, depth first strategy means that the left hand side or "if" side of a rule that contains the most recently asserted fact is at the top of the agenda list, which is the list of rules whose left hand sides are satisfied. The breadth first search works the opposite, it executes the rule whose left hand side uses asserted facts that have been asserted earliest. The simplicity strategy is merely to execute next the rule that has the least amount of statements on the left hand side, while the complexity strategy executes the rule with the most statements on the left hand side of the rule. The lex strategy uses the depth first search, but also uses complexity to break ties. The computer can also be set to execute the rules in a random order. The final strategy is to set the "salience" of each rule by assigning it a value indicating priority and the software fires the rule with the highest salience.(Ref. 12) If the salience is set to control the program, then procedural code could just as easily perform the same function.

B. DEPTH FIRST SEARCH

In depth first search, the search begins at the top layer and proceeds down until the search can not continue any further down. At this point, the search backtracks until there is another possible path that goes down. The search then continues down again until once again it cannot continue any further down. The search then backtracks again, until another path down is found, and continues until all paths have been searched. The depth first search strategy is shown in Figure 4.1

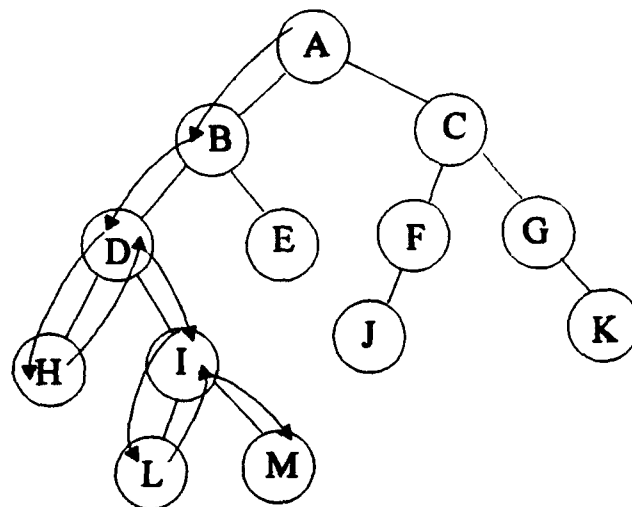


Figure 4.1 Depth First Search Strategy

The depth first strategy can also be explained by stepping through the process as the computational engine would. Suppose the CLIPS rules were defined as

If Someone is a man, then he is male.
 If someone is male, then he can be a father.
 If someone is male, then he is not female.
 If someone is a man, then he is an animal.
 If someone is an animal, then he must eat.
 If someone is an animal, then he is mortal.

Assume that the fact is *Charles is a man*.

This simple program could be implemented in CLIPS as

```
(defrule man
(man ?person)=>
(assert (male ?person))
)
(defrule father
(male ?person)=>
(assert (can_be_father ?person))
)
(defrule not_female
(male ?person)=>
(assert (not_female ?person))
)
(defrule animal
(male ?person)=>
```

```

(assert (animal ?person))
)
(defrule eat
(animal ?person)=>
(assert (must_eat ?person))
)
(defrule mortal
(animal ?person)=>
(assert (mortal ?person ))
)

```

In this case, for a depth first search, the first fact asserted is (*male Charles*). In the next layer down, the fact (*can_be_father Charles*) is asserted. Since the search can go no deeper, then the search goes back up to the male assertion, and then the fact (*not_female Charles*) is asserted. Once again, the search can go no deeper, and must backtrack all the way to the top, and the next fact asserted is (*animal Charles*). Searching under that assertion, then the fact (*must_eat Charles*) is asserted, followed by (*mortal Charles*).

C. BREADTH FIRST SEARCH

In the breadth first strategy, the search starts at the top level. It then proceeds to the next level and evaluates each state at the next level. The search then proceeds to evaluate the next level down under each node previously searched. This continues until all possible paths have been searched. This search pattern is shown in Figure 4-2.

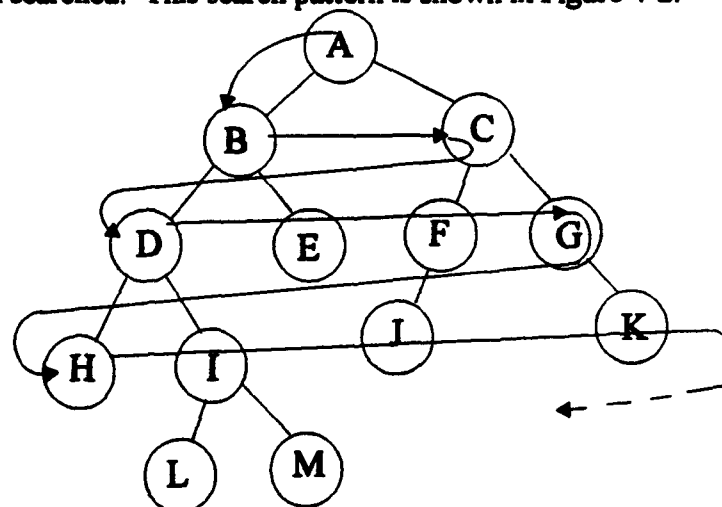


Figure 4-2 Breadth First Search Strategy

The breadth first strategy can also be explained by carrying out the processes of the computer with the previous set of rules about *Charles*. In the first step, (*male Charles*) is asserted. The next iteration carried out is (*animal Charles*). Now all paths to the next layer under the fact that (*man Charles*) have been explored, so the paths from these assertions are then followed. Therefore, (*can_be_father Charles*) is asserted, followed by (*not_female Charles*). Then (*must_eat Charles*) and (*mortal Charles*) are asserted. Even though in this case, the facts asserted were the same, the assertions occurred in a different order. This example shows that either strategy can search all paths, but not all programs can be solved regardless of search strategy. When CLIPS is loaded, the default strategy is depth first.

For the program rules to analyze the telemetry from PANSAT, either search strategy can be used. By pre-determining the order in which key rules perform their operations, the program will be able to do searches on the data without complications from reading in the telemetry data, which is a procedural process.

There are two reasons why the data reading routines are forced to be procedural. When the depth first strategy was used, the first rule in the agenda list was always the "read-data" rule. This caused the program to read into memory all of the data in the telemetry file, and assert those data points as facts. Since the telemetry file is large, the computer quickly runs out of RAM and must store the information on the hard disk, which significantly slows down the computation. By using the breadth first strategy, each data point can be read in singly. Each point is analyzed, and only significant information is retained. The data point is then retracted from the fact list, and the next data point is read into memory. In this way, the least amount of facts are maintained in the fact list, and computation time is significantly reduced. Even though the program was written to have only one data point read at a time, the rules still did not properly work until the rule for reading in the next data point and deleting the old data did not fire unless it was the only rule on the agenda list. By setting the salience on "read-data", the program can be run in either the depth first or breadth first search.

IV. ORBITAL DYNAMICS

The understanding of orbital mechanics is essential to the applications of satellites in space. Every orbit has advantages and disadvantages over other choices. By designing a spacecraft around the orbit that will be used, engineers can benefit from the advantages of space while minimizing the negative effects of the orbit. The orbit has several profound effects on the design of the ground station, and the collection of telemetry. The distance the ground station has to communicate is directly related to the power the transmitter uses. The period of the orbit has a direct impact on the subsystems of the spacecraft. When the spacecraft enters an eclipse, it will switch from solar power to batteries. The satellite will also be hidden from the heat of the sun during eclipse, so the period of temperature fluctuations will vary with the orbit period. The orbit also determines the maximum amount of time the satellite will be in view of the ground station. The higher the altitude, the longer the satellite will be in view. The time the satellite is in view then determines the amount of data that can be transferred to and from the satellite. The length of the eclipse also determines the battery parameters. The eclipse time varies with the relationship between the sun and the orbital plane, and the satellite will be designed for the worst case when the satellite is in eclipse the longest per orbit. In this situation, the energy charged to the batteries during the sunlit time must be at least the same as that used during the eclipse, or the batteries will drain.

A. ORBITAL THEORY

Even though the orbit remains in a plane, its relationship to the Earth can be oriented in any direction. Because of this, we need a reference system that relates the orbit and plane of the satellite to the Geocentric Inertial reference frame as shown in Figure 5-1. The system that is used by the Air Force is called the Perifocal coordinate system. The plane of the system lies in the plane of the orbit. The X direction points at

the perigee of the satellite, the Y axis is in the orbital plane, 90 degrees to the X axis, and the Z axis lies in the direction of the angular momentum vector, \vec{h} .

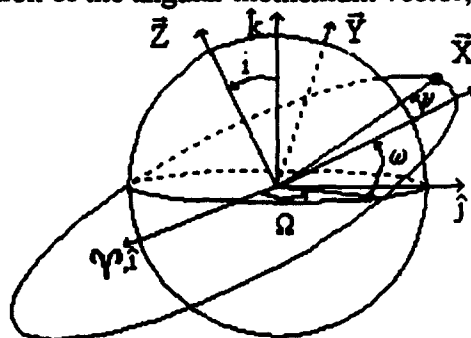


Figure 5-1. Orbital Elements

Fundamental Elements:

Inclination, i - orients the plane in the inertial space. It is the angle between the angular momentum vector and the North Pole, or between the equatorial plane and the orbital plane.

Right Ascension of the Ascending node, Ω - orients the plane in inertial space, the angle is measured eastward from the Vernal Equinox to the ascending intersection of the orbital plane and the equatorial plane.

Argument of Perigee, ω - orients the orbit in the orbital plane, the angle measured from the ascending node to the perigee of the orbit.

True Anomaly, ν - locates the satellite in the orbit, the angle measured in the direction of travel from the perigee to the location of the satellite.

B. NEWTON'S LAWS

The motion of all objects in the universe is governed by Newton's three laws, including the orbits of planets and satellites. His second law, force is equal to the rate of change of momentum, or

$$\vec{F} = m\vec{a}$$

is the starting point for explaining orbital motion. For the gravitational force, the direction of the force is towards the other object, this shall be in the \vec{r} direction as shown in Figure 5-2.

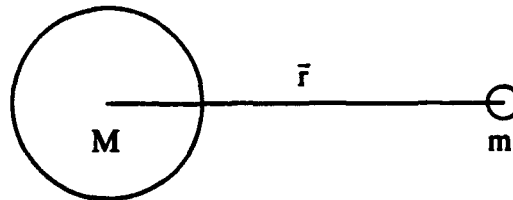


Figure 5-2. Gravity Field

Newton's Law of Gravity states that the force between two objects is proportional to the mass of the two objects, divided by the square of the distance between them, or

$$m\ddot{\vec{r}} = -\frac{GMm}{r^2} \left(\frac{\vec{r}}{|\vec{r}|} \right), \text{ or } \ddot{\vec{r}} = -\frac{\mu}{r^2} \left(\frac{\vec{r}}{|\vec{r}|} \right)$$

where $\mu=GM$. G is the universal gravitational constant $6.672 \times 10^{-11} \text{ m}^3/\text{kgs}^2$. For the sun, μ is $1.327 \times 10^{20} \text{ m}^3/\text{s}^2$, while for the Earth, μ is $3.986 \times 10^{14} \text{ m}^3/\text{s}^2$.

By balancing the centripetal acceleration of the satellite and the force of gravity, the velocity of the satellite in orbit can be determined. According to Kepler's third law, the period of the orbit is

$$T = \frac{2\pi}{\sqrt{\mu}} a^{3/2}$$

The force of gravity must be equal to the centripetal acceleration in order for the satellite to be in a circular orbit, so the equation becomes

$$F = ma = \frac{\mu m}{r^2} = \frac{mv^2}{r}$$

Solving for v yields

$$v = \sqrt{\frac{\mu}{r}}$$

This shows that for a higher orbit, the velocity is lower. (Ref. 13)

C. TWO LINE KEPLER ELEMENTS

One of the many systems used to transmit the orbital information on a satellite is to use the 2 line element Kepler system. This is the system used by most bulletin board systems that download satellite ephemeris, and is used to update the files in the ground station's tracking program InstaTrack. An example of the information in the data files is

```
E +.11846032+01 +.37358470-02 +.79893359+02 +.90024519+02 30490 1 94 1  
E +.44748849+02 +.71002327+02 -.27310065+01 +.13061904-02 30490 1 94 2
```

The first number on the first line is the semi-major axis of the orbit measured in Earth radii. The next number is the eccentricity. After that is the epoch of perigee which is measured in kiloseconds from midnight. The last number on the first line is the orbital inclination, measured in degrees. The next field is the satellite number. After that is the date, followed by the card or line number. The first number on the second line is the argument of perigee, ω , measured in degrees. The next number is the right ascension of the ascending node, Ω , also measured in degrees. The third number is the rate of change of the argument of perigee, measured in degrees per day. The last number of this line is the rate of change of the right ascension of the ascending node, also measured in degrees per day. The last set of numbers has the same meaning as for the first line.

D. ORBITAL PERTURBATIONS

If there were no forces besides gravity centered at the center of mass of the Earth, then the satellite would stay in the same orientation for all time. Because of many factors, this does not happen. The effects of the Sun, Moon, and all the planets, drag, solar radiation pressure, and mass asymmetry cause the orbital plane and the shape of the orbit to constantly change. The perturbations are any forces acting on the satellite which are not the uniform gravitational force of the Earth.

The Earth is not a perfect sphere, but it is slightly flattened, making a bulge at the equator. The northern polar region is also flatter than the southern, making a slight pear

shape. The equator isn't perfectly round either, but is slightly elliptical. The primary effect of the asymmetrical mass of the Earth on a low orbiting circular satellite is changes in Ω .

The rate of change of the ascending node is a function of the altitude and eccentricity of the orbit, but is mostly affected by the inclination. An inclination of 90 degrees results in no change in the ascending node. Figure 5-3 is a Mathcad plot showing the relationship between the rate of change of the argument of the ascending node, the inclination, and altitude. (Ref. 13) As the inclination decreases, the rate of change increases westward until a maximum at a zero degree inclination. For a retrograde orbit, the increase in inclination results in an increase in the rate of change to the East. For inclinations greater than 90 degrees, use the positive value of the inclination minus 90 degrees.

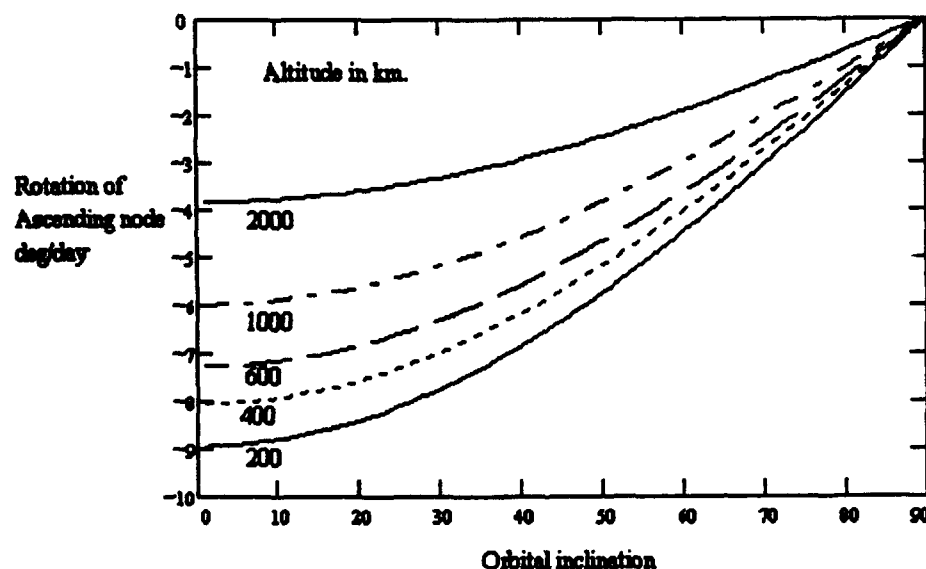


Figure 5-3. Orbit Precession with Respect to Orbital Inclination and Altitude

Everything in the universe exerts a gravitational force on the satellite, but most are negligibly small. For satellites in a high orbit, like geosynchronous, the effects of the sun and the moon start to become noticeable. These mostly affect the right ascension of the ascending node and the argument of perigee and the forces are about five to 100 times weaker than the perturbing force of the Earth's oblateness. In order to make corrections for multi-body problem, the sun and moon are assumed to orbit the Earth, and an iterative

process is used to find the satellite at certain time intervals. This process is very time consuming and not feasible for accurately tracking large numbers of satellites.

The physical characteristics of the launch site also place constraints on the inclination of the low earth orbit of the satellite. Launches from Cape Canaveral must be at least 28.7 degrees because of the latitude of the launching pad. To the North of the launch site there are heavily populated areas, consequently the upper limit of the inclination is around 70 degrees. When satellites are launched from Vandenburg, they have to go into a retrograde or polar orbit because of the land constraints around the base.

E. LOW EARTH ORBITS

Many satellites are put into a low earth orbit because for varied reasons. The fuel required to put a satellite into a geosynchronous orbit make the mission very expensive, while putting a satellite into a lower altitude saves the added weight and structure needed with the more fuel required. At geosynchronous orbits, the beam width of an antenna or other device must be very narrow to get good resolution of the Earth from the height of the orbit. In a low earth orbit, a larger beam width can be accepted for the satellite, or much greater resolution can be attained with the same instrument. The main disadvantages of this orbit are the increased atmospheric drag and gravitational anomalies that are stronger at the lower altitudes.

F. DETERMINATION OF TIME IN VIEW

The time that a satellite is in view can be determined from simple geometry, assuming that the orbit is circular and that the ground station is in the same plane as the orbit. As shown in Figure 5-4, a triangle can be made with vertices at the center of the Earth, the ground station, and the satellite position when it is E radians above the horizon. Two of the sides of the triangle are known, the radius of the Earth, and the radius of the orbit. The FCC rules state that a satellite link cannot be made less than 10 degrees above the horizon, so this will be used as E .

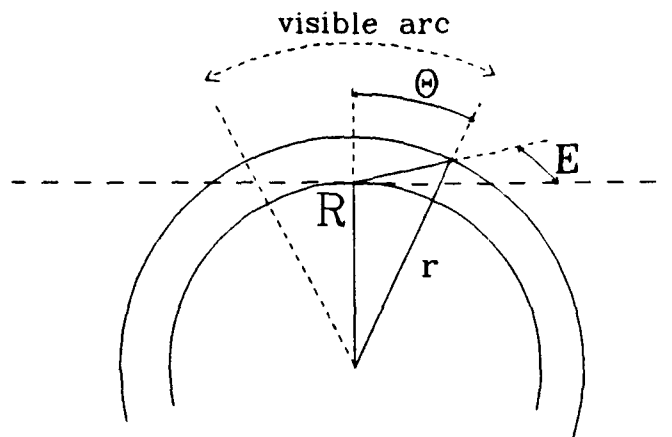


Figure 5-4, Determination of Time in View

From the law of sines,

$$\frac{\sin(\pi/2 - \theta - E)}{R_e} = \frac{\sin(\pi/2 + E)}{r}$$

The only unknown in the equation is θ , which can be solved for. The total angle in view is $2 \times \theta$. Once the total period is known, and with θ in radians to give the partial fraction of 2π radians for an orbit, then the time in view is

$$T_{\text{view}} = \frac{\theta}{\pi} \times T$$

For the PANSAT requested altitude of 450 km., the maximum time in view is 8.6 minutes. The time in view is a function of both the arc distance that the satellite passes through, and the rate at which it traverses overhead. The relationship between time in view and orbit altitude for an overhead pass is shown in Figure 5-5 which is determined using Matlab. As the altitude increases, the arc traveled increases slightly, but the greatest contribution to increased time in view is the longer period.

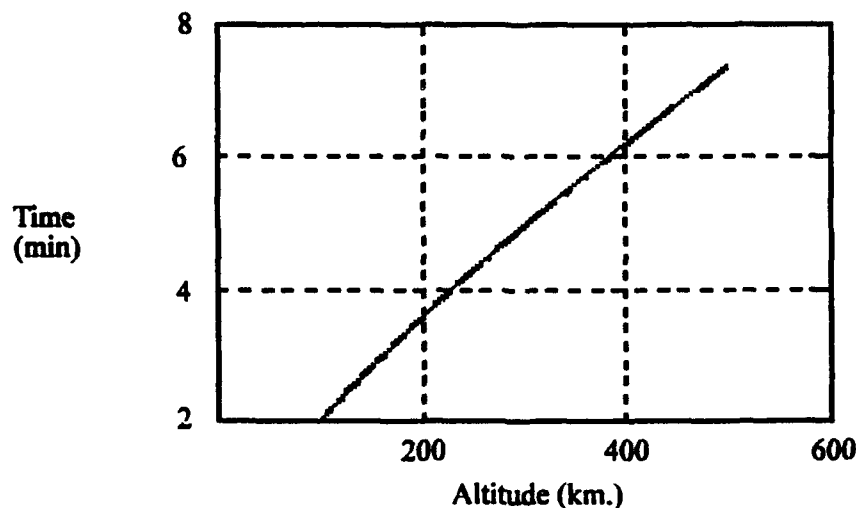


Figure 5-5, Plot of Time in View vs. Altitude

In general, there are two to three passes in a row of a low earth orbiter, then the next pass is about ten hours later as the station rotates to under the opposite node. If one pass is near the maximum time overhead, the other two are significantly shorter. If the inclination is only 28.7 degrees, then a ground station in Monterey, CA may only have one period daily when communications are possible.

G. DETERMINATION OF TIME IN ECLIPSE

Since PANSAT will be in a low earth orbit, it will enter eclipse on every orbit. The eclipse is caused by the satellite entering the shadow where the Earth blocks the sun, as shown in Figure 5-6. The duration of the eclipse constantly changes with the seasons since the declination of the sun moves from plus to minus 23.45 degrees and orbit plane orientation as it precesses.

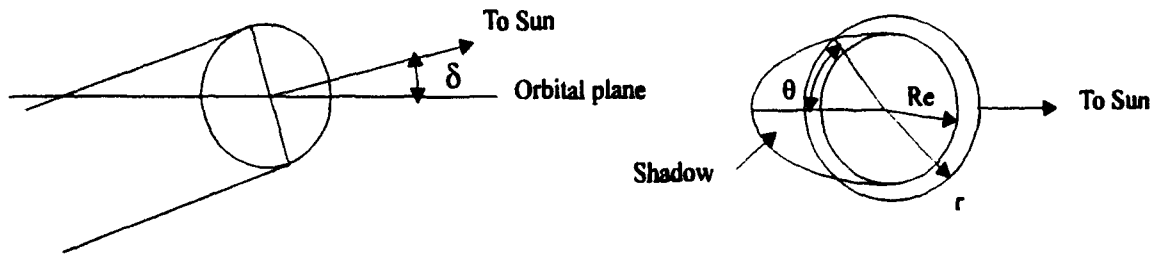


Figure 5-6, Earth Shadow Intersecting Orbital Plane

This change in the angle between the sun and the plane of the orbit determines the time of eclipse. The intersection of the Earth's shadow and the plane of the orbit make another ellipse, whose equation (Ref. 11) is

$$\left(\frac{x}{R_e / \sin \delta} \right)^2 + \left(\frac{y}{R_e} \right)^2 = 1$$

Since $x = r \cos \theta$ and $y = r \sin \theta$, then the equation becomes

$$\theta = \cos^{-1} \left(\frac{[1 - R_e^2 / r^2]^{1/2}}{\cos \delta} \right).$$

To relate the angle θ to the period, it must be found in radians. Then the total eclipse period can be found as

$$T_e = \frac{T \times \theta}{\pi}.$$

The eclipse time is the maximum when the sun is in the orbital plane, or $\delta = 0$ degrees. (Ref. 14) For the PANSAT in a nominal Space Shuttle orbit, this relates to an maximum eclipse period of 35.9 minutes. The maximum percent time in eclipse is then 38.4%.

H. SIGNIFICANCE TO GROUND STATION AND A. I. ROUTINES

The telemetry analysis will use only the telemetry data stream to make its decisions. The satellite events will be compared with the time the satellite leaves eclipse, and reenters sunlight as determined by the telemetry itself. This comparison will determine if there is a correlation between the fault and the orbit.

The ground station must also know the ephemeris of the satellite in order to track it. Since the system is open loop, the antenna points to the predicted position. If the ephemeris is incorrect, or the ground station's time is off, then the antenna will not be pointing at the satellite. The time is also very important in determining the Doppler shift of the satellite. If the time is off by only 20 seconds, then the Doppler shift will be off by 400 Hz. This could cause the system to not link. The ephemeris of the satellite is constantly changing due to unaccounted perturbations in the gravitational field, and drag not accounted for in the 2 line ephemeris system. The data will need to be updated regularly for the ground system to track PANSAT.

V. THERMAL CONTROL

The heat input to a spacecraft varies widely in the space environment. When the sun lights on an object in space, the only method to transfer heat is in the form of radiation. This solar flux all but disappears when the satellite enters the Earth's shadow and only reflections and emissions from the Earth and moon affect the heat transfer. Since the solar flux is approximately 1358 W per square meter, the satellite must be able to operate in both extremes, from zero to maximum flux. In the sunlight, the temperature increases significantly as the spacecraft absorbs the sun's radiation. In eclipse, the temperature decreases rapidly as the satellite radiates energy to space. The thermal properties of the satellite must be well known before the satellite is launched to insure correct operation at all possible extremes of the temperature range.

A. LAWS OF THERMAL RADIATION

An explanation of thermal radiation can be found in quantum mechanics. A black body emits radiation at all wavelengths, and is modeled with Planck's Radiation Formula

$$\epsilon = \frac{2\pi hc^2 \lambda^{-5}}{e^{ch/k\lambda T} - 1}$$

where ϵ = power radiated per unit area of the body per wavelength, c = speed of light, $3 \cdot 10^8$ m/sec; h = Planck's constant, $6.62 \cdot 10^{-34}$ J·s; k = Boltzmann's constant, $1.38 \cdot 10^{-23}$ J/K; T = temperature in Kelvins; and λ = wavelength in meters. Knowing the temperature of the emitting body, the total amount of power radiated per unit area can be determined by integrating Planck's Law over all wavelengths, from zero to infinity, or

$$\epsilon_{\text{total}} = \int_0^{\infty} \frac{2\pi hc^2 \lambda^{-5}}{e^{ch/k\lambda T} - 1} d\lambda$$

which results in the Stefan-Boltzmann Law

$$\epsilon_{\text{total}} = \sigma \cdot (T^4 - T_0^4)$$

The constant σ is the Stefan-Boltzmann constant, $5.67 \cdot 10^{-8} \text{ W/m}^2\text{K}^4$. The final product of the equation has units of W/m^2 . The total power radiated by the body is found by multiplying by the total surface area.(Ref. 15)

These laws apply only if the source of radiation is a black body, one that absorbs all energy at all wavelengths, and emits at all wavelengths according to the body's temperature, which is usually not the case in the physical world. The emission difference between the actual body and a black body is wavelength dependent, determined by the actual rate of emission compared to a black body at the same temperature. This comparison is called the monochromatic emissivity, ϵ_λ . The monochromatic absorptivity is calculated similarly, and is denoted by α_λ .(Ref. 15)

The same equations can be used to determine the temperature of the solar cells, which is important in determining the number of cells needed to power the spacecraft since the power varies with temperature. The solar cell determination makes a change to the equations so they will be applicable. The solar array uses some of the energy that is absorbed to make electricity, so it is not converted into heat. This is accounted for by the effective absorptivity as

$$\alpha_{\text{eff}} = \alpha_s - Fp \cdot \eta.$$

In this equation, the Fp is the packing factor of the solar cells on the array, η is the solar cell efficiency, and α_s is the solar absorptivity.(Ref. 14)

The total energy transferred by the emitter is then

$$q = \sigma \cdot s \cdot \epsilon \cdot (T^4 - T_0^4)$$

where s is the surface area of the emitter, and T_0 is the temperature the energy is radiated to, whether it is space or another body. Within the spacecraft, the radiation between two components is also dependent upon the spatial relationship between the emitters and the power dissipated by each component. This will account for the fact that not all energy

emitted by a source can be intercepted by another body. These factors are known as shape factor, arrangement factor, and view factor; Figure 6-1 illustrates these factors.

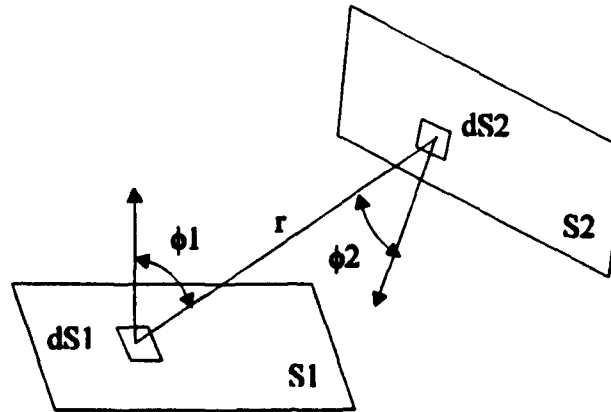


Figure 6-1, View Shape Factor Determination

The shape factor is found from

$$\int \int_{S1 S2} \frac{\cos \phi 1 \cdot \cos \phi 2 \cdot dS1 \cdot dS2}{\pi r^2} = S1 \cdot F_{A12}$$

and the radiation from one surface to another is then

$$q = \sigma \cdot \epsilon \cdot S1 \cdot F_{A12} \cdot (T_1^4 - T_2^4)$$

The temperatures are those of the respective bodies. The energy radiated from S2 back to S1 can then be found in the same way. (Ref. 15) When the configuration of a spacecraft is known, then this procedure must be used to determine the shape factors, and energy transmitted from each surface to all others within the spacecraft. When all equations are solved for the surface temperature, then the solution is found. There are different methods for determining the final temperature. The first is to use an iterative method, where the temperatures are assumed, and the emittance is found. Then the temperatures are changed to correct for the absorptance from other surfaces, and continues until the steady state temperatures are found. The system of equations can also be set up as a resistor network, where each surface is a node whose energy flow is

represented by a voltage, and the resistance corresponds to the shape factor, the current represents the radiant energy. There are software programs and tables to aid in determining the shape factors and temperatures.(Ref. 15)

If the satellite has an extreme temperature range, there are a couple of solutions to consider in the design. One solution is to design a radiator to emit excess heat to space. The size of the radiator is a function of its properties, and the amount of heat that needs to be dissipated. The ideal radiator would have minimal absorptance and maximum reflectance at its operating temperature. Due to exposure to the space environment, radiators lose some of their ability to emit heat over time, so the worst case for heating the spacecraft happens at the end of life. Heat is created in the spacecraft by electrical resistance. The equation for determining the size of the radiator is

$$A = \frac{P}{\eta \cdot \epsilon \cdot \sigma \cdot T^4 - \alpha \cdot S \cdot \sin(\delta)}$$

where P is the power is to be dissipated from the spacecraft, η is the efficiency of the radiator due to reflections off structures, and T is the highest tolerable temperature in the spacecraft design, S is the solar heat input, and δ is the incidence angle. The rest of the spacecraft is covered with insulation to prevent either the gain or loss of heat which is not be modeled or compensated for. When the radiator is not facing the sun, then the temperature can be found as

$$T = [P / \eta \cdot \epsilon \cdot \sigma \cdot A]^{1/4}.$$

In eclipse, the lowest possible temperature is found with η equal to 1.0.(Ref. 14)

Another solution to change the temperature range is to add heaters to the satellite. In this situation, the satellite draws energy either from the solar cells, or the batteries to power heaters, which convert the energy into additional heat dissipation. This translates into a higher spacecraft temperature.

B. THERMAL CONTROL OF PANSAT

In determining the thermal characteristics of PANSAT, there are several factors that are a given. The heat input of the sun is well known, as well as the albedo of the Earth and moon. The solar panels which cover the majority of the spacecraft also have a known emissivity and absorptivity. Once the spacecraft configuration and components have been determined, and the process of finding the thermal balance of the system can begin.

Due to the limited power margin, the thermal control of the spacecraft will be passive. The solar cells and the open corners will absorb the heat as well as transmit heat to space. The major energy dissipation will come from the batteries, the digital computer, and the communications receivers and transmitters.

Once a thermal model of the spacecraft can be programmed, then the operational limits of the components of the spacecraft will be known. The thermal range of the satellite will be well modeled and tested before launch. From preliminary analysis performed by Professor Kraus of the Naval Postgraduate School, the satellite will not have a problem with overheating.

C. THERMAL DECISIONS

The Artificial Intelligence decisions of the thermal control system will only be able to check on the operation of the system since it is totally passive. The only possible corrections for a passive system once it is in orbit will be either to shut down systems to cool off the spacecraft, or to use more of the systems to increase the temperature. In PANSAT, the only system that can be controlled to have an effect on the temperature of the satellite will be the communications subsystem. If the transmitter gets too hot, then the Digital Control System logic may be set up to not transmit. This will cause the energy to be dissipated in other subsystems. Another possibility is that the attenuation level on the transmitter can be increased, so not as much energy is transmitted, and therefore, less heat is generated. On the other hand, if it gets too cold, it can constantly transmit to heat up the spacecraft. This solution runs into the problem of the spacecraft not having enough

power margin in the batteries to allow constant transmissions throughout an eclipse. The redundancy of systems on board can also help in the thermal control. If a subsystem is having temperature problems, the alternate can be selected which does not appear to have the same dilemma. Operations can also affect the temperature of the batteries. If the batteries are overheating, and they are being fully charged, then a trickle charge could be used. Instead of possible overcharging the batteries, they will remain charged, but won't convert excess energy into heat.

VI. POWER SYSTEM

The design of the power system for PANSAT has taken a different course than for the typical satellite. Rather than the payload determining the amount of power that is required of the system, the physical dimensions and design of the satellite restrict the size of the solar panels, and thereby determine the total power available. The satellite will have 17 solar panels, each 7.125 inches square, capable of producing 6.12 Watts each at 15.82 volts at AM0 and 28° C.(Ref. 16) After much deliberation, the power storage will use X-rayed and intensely inspected commercial Nickel Cadmium (NiCad) batteries instead of space qualified cells. This greatly reduces the total cost of the power system, and has proven effective in other amateur satellite designs.

The power constraints on the satellite have left very little margin for error. The power system is also generally the one that gives the satellites the most problems if they are poorly designed. The success of the entire satellite rests upon the power that it can produce, and fluctuations can cause havoc in other subsystems. The power system must be able to switch from external solar power to internal battery power without major power spikes, or fluctuations. Since the period is on the order of 90 minutes, the power system will be continually cycling between all modes of operation.

A. SOLAR ARRAY DESIGN

In designing the solar arrays, PANSAT is limited by the size of its panels that the solar cells will be mounted on. The panels are designed by Spectrolab, Inc., and meet the specifications determined by the Space Systems Academic Group. Since a low Earth orbit is not a harsh radiation environment, solar cells do not need to have exceptional resistance to radiation. The total radiation dose over the lifetime of the satellite as determined by Spectrolab Inc. is a fluence of 1.80×10^{13} 1 MeV equivalent electrons. A fused silica coverslide of 30 mils thick will block all of the high energy protons and offer added

protection during construction, testing, and launch operations. The cell chosen is a Series 6700, back surface field, back surface reflector, 10 ohm which is 1.92 cm. by 4.00 cm. At the beginning of life, (BOL), the cell characteristics are open circuit voltage (Voc) 605 mV, and short circuit current (Isc) 319 mA. There are several factors that degrade the solar cells over the lifetime of the satellite. The first factor is a darkening of deposits on the fused silica coverslip and the adhesive used to attach the coverslip to the solar cell. This causes a degradation of the cell short circuit current less than 1%. On short duration missions, Spectrolab assumes micrometeorites cause little or no power loss so the total loss was assumed to be .98% for both factors. The radiation also causes reduced efficiency within the solar cells. Over the lifetime of the satellite, the predicted end of life (EOL) values are predicted to be .965 of the BOL volts at maximum power (Vmp), and .978 of the BOL amps at maximum power (Imp). There is also a panel wiring loss of .017 volts which reduces the usable power from the solar cells. Taking all of these factors into account makes the EOL performance 21.67 Watts at 15.27 volts, when the satellite is in the most favorable attitude.(Ref. 16) The current of a solar cell falls off as the cosine of the angle the sun makes perpendicular to that cell, which in turn makes the panel power drop. The four sides are 45° off of the side facing the sun, so the power of each is $\frac{\sqrt{2}}{2}$ times the power of the side facing the sun. Adding all sides together, the total power is then $\left(1 + \frac{\sqrt{2}}{2}\right)$ times the maximum power of one side. The EOL values assume summer solstice solar flux, 28° C and 90° incidence angle to one of the faces with four adjacent panels being partially lit. The solar panel power is reduced by diode losses before the satellite can use that power on the spacecraft electrical bus.

The solar array temperature found in the thermal control analysis, with the temperature coefficient for current, α_I , and voltage, α_V , are used to verify the panel characteristics. With these factors, the cell current at the EOL can be found using

$$I = [I_{mp} + \alpha_I \cdot (T - 25)] \cdot K_A \cdot K_D \cdot K_r.$$

In this equation, T is the temperature of the cells, K_A is the design factor for assembly losses in current, K_D is the factor for environmental degradation, and K_S is the solar intensity factor. The temperature of the cells will be the lowest when the satellite leaves eclipse, so this is when the high power point can be expected. The voltage requirements of the bus must also be met. The voltage per cell is given by

$$V = [V_{mp} - \Delta V + \alpha_v \cdot (T - 25)] \cdot K_E$$

where ΔV is the panel wiring loss per cell, and K_E is the radiation degradation. The number of cells in series can then be found by

$$N_s = \frac{\text{bus voltage} + \text{bus voltage drop}}{\text{cell voltage}}$$

The solar panels are then connected in parallel to the solar bus, which powers the electrical system bus and charges the batteries. The same equations are used to insure that the power requirement will be met during the winter solstice when the solar flux is higher, which raises the cell temperature, which has an effect on the total power. (Ref. 14)

B. DESIGN OF BATTERY

The design of the battery cells is based upon a solar cell output of 15.27 Volts, and a battery voltage at least 11 volts. The depth of discharge for the batteries, DOD, is an extremely conservative 10%, and the EOL voltage, V_d , for a fully charged battery is expected to be 1.3 volts per cell. According to the equation

$$V_{db} = (N - 1)V_d - V_{dd},$$

the number of cells is 10 when the voltage drop of the bypass diode, V_{dd} is .7 V. At the EOL, the actual bus voltage will be 12.3 volts assuming no cells fail, since $(N-1)$ takes into account that one of the cells may short circuit. The required cell capacity was found by using

$$C = \frac{P \cdot t}{V_{db} \cdot DOD},$$

where P is the power required from the batteries, and t is 36.6 min. for a Shuttle orbit. For PANSAT, 5 amp-hr batteries have been selected.

The next step is to determine the power required to charge the batteries during the solstice and equinox seasons. Using C/15 for charging when in a maximum eclipse orientation, and a current of .33 amps, the power required to charge the batteries is $P_{chg} = \text{current} \cdot \text{voltage}$. Assuming a charging efficiency of 90%, the time to recharge from

$$t_{recharge} = \frac{P \cdot t}{P_{chg} \cdot \eta}$$

is 85.9 minutes. This time to recharge the batteries is more than the time in sunlight per orbit, so the satellite will not be able to operate at full capability. (Ref. 14) By combining the time to recharge equation, with the eclipse and period equations, the relationship between the orbit and the electrical system becomes,

$$\left(1 - \frac{R_e^2}{r^2}\right)^{1/2} = \cos\left(\frac{\pi}{\frac{P}{P_{ch} \cdot \eta} + 1}\right)$$

For the full satellite capabilities, the satellite orbit would have to be 1,602 km. in order for the batteries to be able to recharge fully every orbit. If the orbit is set at 450 km., then the eclipse power consumption would have to be less than 7.28 W.

C. BATTERY CIRCUITS

The battery circuits for operating the power system are controlled by signals from the Digital Control System (DCS), shown in Figure 6-1. When the DCS sends a signal to enable any of the charge circuits, it controls the current through the transistors which then allow current to flow to or from the battery.

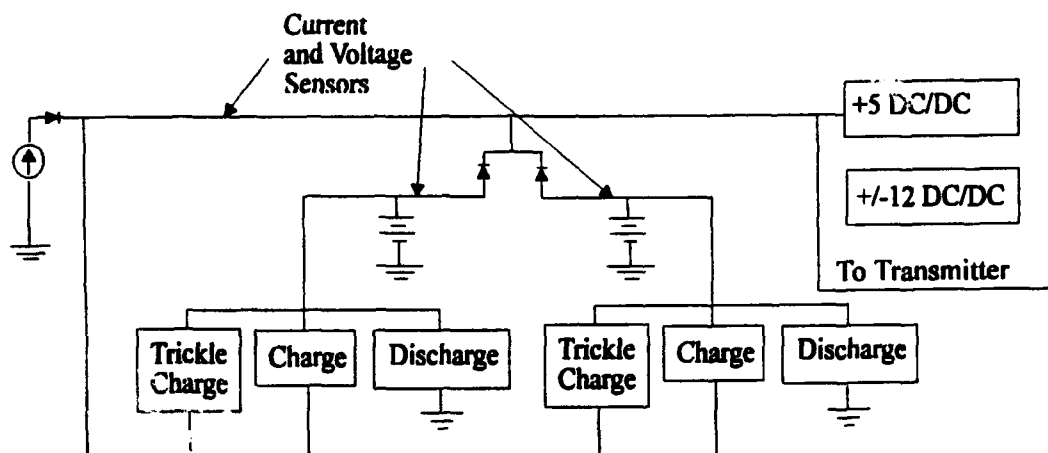


Figure 6-1, Electric Power System Block Diagram

When the satellite is in the sun, the electrical bus voltage will be higher than the battery voltage, so the diodes prevent the batteries from charging directly off the power bus, as can be seen in Figure 6-2. Instead, the DCS senses the power voltage and current, and determines that the satellite is in the sun, and the batteries can be charged. When the DCS notices that the bus voltage is lower than the power bus, then the batteries are powering the electrical bus, and the charge circuitry is commanded "off".

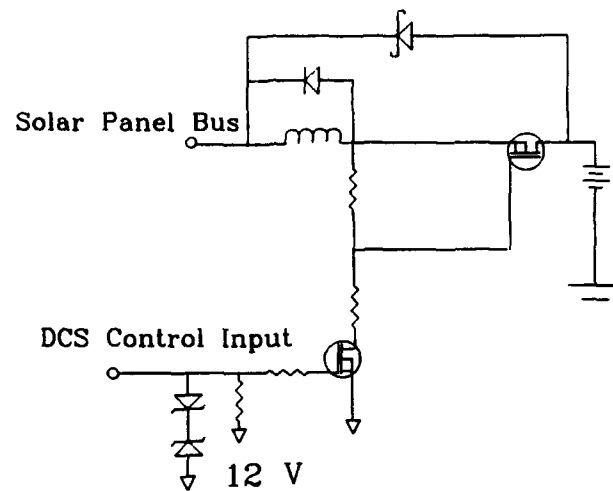


Figure 6-2. Battery Charging Circuit

The battery charging system also has a trickle charge capability shown in Figure 6-3. In the present design, this will only be used when the cells are completely dead. If the batteries have no or very little voltage, as is expected when the satellite is jettisoned from the launch platform, a fast charge can damage the batteries. The trickle charge is used to charge the batteries until they can be safely charged at the full rate.

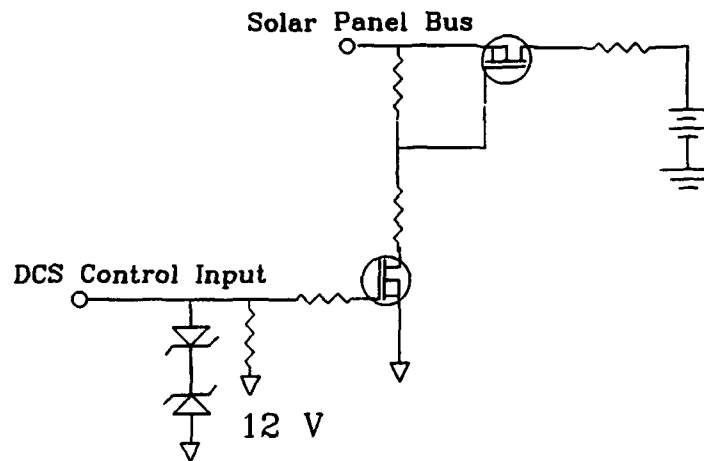


Figure 6-3. Battery Trickle Charge Circuit

The batteries must also be periodically drained to reduce the memory effect that plague NiCad batteries. If the batteries are not reconditioned, then they eventually lose

the ability to fully charge. This is prevented by draining the batteries through resistors to ground, see Figure 6-4. Since the batteries are designed so that each one can power the spacecraft during eclipse, the cell reconditioning can occur over several orbits and one set of cells is reconditioned at a time. The interval to recondition the batteries is a function of the depth of discharge, and the information is provided by the supplier.

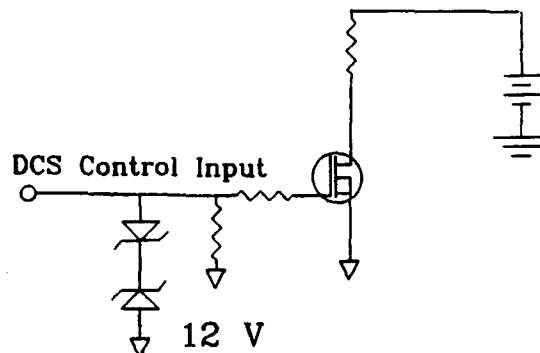


Figure 6-4. Battery Discharge Circuit

There may also be problems during the sunlight operations. There is a significant drain on the power system when the communications system is working. If the solar cells are not producing enough power run the satellite and transmit, then the batteries will supply the extra current needed to operate the system.

There are a couple of possible reasons why the spacecraft would operate in the negative power margin region. The first is that the bottom panel without a solar panel may be predominantly facing the sun. In this attitude, the spacecraft receives significantly less power than for which the system is designed. The power system design assumes that at least the average overall power is available at all times. Almost any orientation and spin axis that the satellite takes on after ejection will have the possibility that the power margin will be negative at some time. If in sunlight, the satellite will also probably be negative on the power budget only during transmission. Using all satellite capabilities during eclipse draws more power than can be recharged during the next sunlit period, so the operational characteristics of the satellite must be reduced account for this power loss.

Depending upon the severity, and the number of occurrences of power problems, the power budget could be a major obstacle, or a minor inconvenience. If the power budget is not significantly draining the batteries during the charge cycle, then there may be no corrective action needed. It could very well be that the satellite only goes into a negative power budget when the blank panels is facing the sun, and the satellite is transmitting. Depending upon the attitude of the spacecraft, and the duty cycle of the communications, this may occur very seldom. On the other hand, the satellite could have a spin axis and attitude where its bare panel faces the sun during a significant part of the orbit. In this case, drastic measures may have to be taken.

Since the satellite can sample and read the solar cell telemetry at will, the satellite could only transmit when it reads enough current from the solar cells to power the transmitting spacecraft. Another conclusion could be to not transmit at night, during eclipse. This would keep the batteries as highly charged as possible, so that they could supplement the solar panels during daytime transmissions.

From the manufacturers documentation, the relationship between battery voltage, temperature, and load can be determined when the battery is fully charged. Since the load on the batteries can be determined from battery current telemetry, and the voltage and temperature are also known, it can be determined if the batteries are fully charged. By linearizing the suppliers plots of voltage and temperature, the equation of a line will determine if the batteries are at or above the voltage threshold for a given temperature. If so, the rule determines that the batteries are fully charged.

There are a couple reasons why the battery voltage may be too low. As the battery drains, and its temperature changes, the voltage will vary constantly. The lower the temperature, and more drained the battery is, the lower the voltage will be. If the power margin is such that the batteries are not fully charged as the satellite enters eclipse, then as the batteries drain, their voltage may drop below limits. The batteries also have a memory effect so that they do not work as well after they have been used and not completely discharged.

A high battery voltage must be caused by the spacecraft power system since a battery's voltage is limited by the potential of the chemical reaction within the battery. The charging circuit on the satellite uses the solar cell bus to supply excess power to the batteries for charging. The electrical power system will charge the batteries at a full charging rate at all times. It is expected that with the satellite's low power margin that the current flow will be low and voltage within limits.

Depending upon the spacecraft's attitude, it may receive more power than expected, which could in turn lead to excess heat in the batteries. The battery temperature is closely tied with the battery voltage. As in the high voltage case, a high temperature could be caused by over charging the batteries, or charging them too fast.

The bus voltage is directly tied to the battery charging rate, so if the bus voltage is out of limits, the satellite should trickle charge the battery to limit the charging current flow. A high transmitter bus voltage is also directly tied to the electrical power system bus, so there is most likely a problem in the overall power system.

When the satellite first comes back into the sun, the solar cells are at their lowest temperature, and highest voltage. As the cells heat up, they do not produce as much power.

VII. SATELLITE CONFIGURATION

The satellite has several redundant subsystems, shown in Figure 7-1, so the overall probability of a successful mission life are greatly increased. The satellite has only one power source, the solar cells, which would limit or end the satellite operational life if problems occur. The power can be stored in either battery since each can operate the satellite individually if need be. There will also be two Digital Control Systems (DCS), multiplexers, mass storage units, transmitters, and radio frequency sections. There is only one antenna system, which is another area of single point failure.

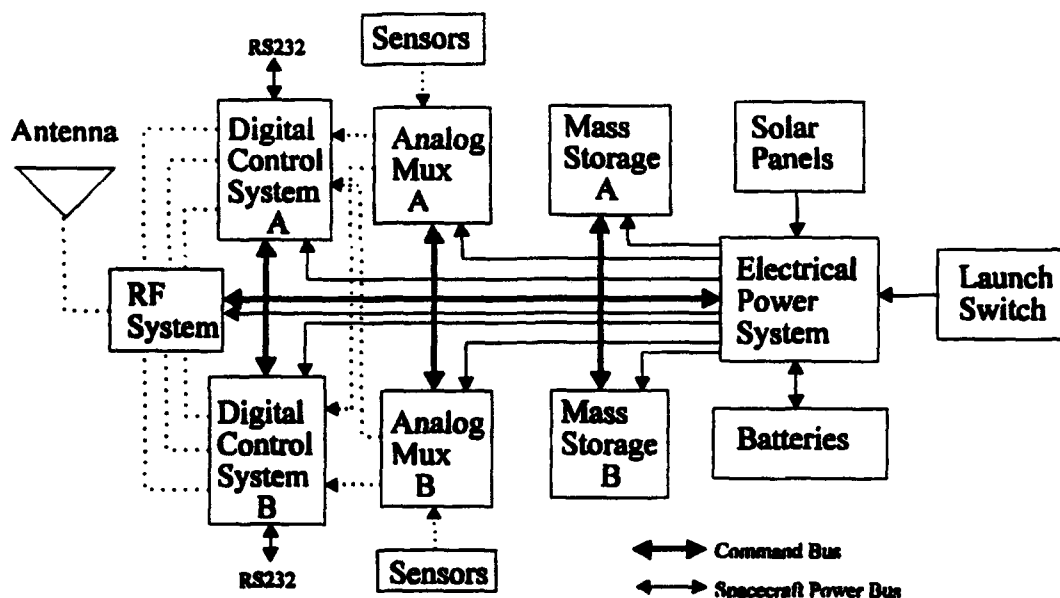


Figure 7-1, Satellite Configuration Block Diagram

The satellite configuration will be a major aid in determining satellite problems. All major subsystems can be switched to isolate faulty boards. If any fault is detected that cannot be corrected, or adjusted, then the system can be switched. Even the DCS can be switched without losing the telemetry file in the mass storage unit, so the ground can verify the satellite's decision. The ground station will have very little control over board

temperatures and power consumption. If a temperature limit in the transmitter board is exceeded, the power out and amount of transmissions can have an impact on the temperature readings. The other subsystems do not have that luxury, and will have to be switched before they cause system failure.

VIII. CLIPS TELEMETRY ANALYSIS RULE SET

The CLIPS rule set for analyzing the telemetry file can be divided into four categories of rules. The first set of rules reads all files that the program uses, and changes the data into asserted facts. The second set of rules does limit checking on the telemetry values, while the third set is the operational rules to determine if the satellite is properly operating. The last set of rules writes all knowledge about the facts into output files, which can then be read by other routines and displayed to the user.

The first set of rules currently reads in three files. The first two are the stored telemetry file, and the current telemetry file. These two files are exactly the same in format, the only difference is that the stored telemetry file contains all data recorded since the last telemetry download, so there is significantly more data than in the current telemetry file, which only contains the most recent telemetry reading. The format of the telemetry needs to be converted from the binary format that it comes from the satellite into an ASCII data file that the CLIPS software can interpret. Each telemetry reading will be divided into sets of data. The first line in the telemetry set must be the time tag for the following data. All other data can be in any order and the rules will still work properly, which is one of the advantages of A.I. All lines must also be in a format that the program can read in logically. The first field is the telemetry point name. The second field is the number of data values that this telemetry point has. The rest of the line is the data points associated with that telemetry point. The number of data readings must equal the second field, or the program will not work properly. Each field and data point is separated by a space. The first couple of lines of a telemetry file may be as follows.

```
time 1 41564
batcur 2 10 10
cellcur 1 0
cell 17 25 30 25 25 30 -25 -17 -30 25 25 80 125 25 30 25 25 30
```


The first line is the time tag. There is only one time, so the next field is "1", followed by the time the telemetry was read. There are two battery current reading ;, so after the telemetry point name "batcur" is a "2", followed by the two measurements. The same format continues through the file. When a data point is read, it is asserted as (tel subsystem sensor_number value time), which is used by all other rules.

The second file to be read is the configuration file. This file contains the configuration that the ground station thinks the satellite is in. This file's format is as follows.

```
rxset 1
txset 2
modeset 1
attenset 4
dcsset 1
verset 4
```

The format for this file is simply the subsystem, and the one operating. This file may be in any order, as long as each line follows the same format. Upon launch, this file will be in the default configuration which the satellite should power up in. Whenever the spacecraft is commanded to change subsystems, this file will be updated to reflect the commanded change.

The satellite will have two more files that have not yet been fully defined. The first is the Event Logger File. This file will contain the information that the satellite decided is pertinent as an anomalous event which requires action. The commands that the satellite has received or decided itself and executed will be stored in the Command Logger File. When the parameters from this file are compared with the telemetry, the software will decide if there has been a change in configuration, and then determine if it agrees with the satellite's decisions through analysis of the telemetry. There will be two more routines developed to read in the event logger and the command log when the two systems are better defined.

The limit checking routines determine if the telemetry values are outside of the expected operating ranges. If a point does lie outside of the expected parameters, then the

rules will make assertions concerning the fact, offer corrective action, and determine the telemetry color that should be displayed to the user. The rules take the worst case, and only present those color schemes to the user at the top level. The format for an out of limit reading is (pt subsystem sensor_number problem time). The decisions are stored as facts in (dec decision_made) format. These rules also assert the telemetry subsystem colors as (color subsystem sensor_number red_or_yellow) and (color category red_or_yellow). As rules are added, using these formats will make them compatible with the rest of the rule set.

The next set of rules check on the operations of the satellite. These include determining when in the orbit a point first went out of limits. Another possibility is that a telemetry point is going out of limits not constantly, but during each orbit. The temperatures and voltages fluctuate depending upon the time in the sun. The analysis routines look for the first time a limit is exceeded per orbit, and can determine if the point is staying out of limits at all times. Other rules determine the highest and lowest values of each point in the telemetry file for output to the user, and check on system operations. This should give the user an idea of the long term trends of the satellite data. Even though the data points may not be breaking any limits, by looking at only the highest value per orbit, one can tell of the trend is increasing such that at some time in the future, it will exceed one of the limits. If the data is assumed to be increasing at a linear rate, then the slope of the line can predict when the satellite may exceed its limits. On the other hand, if the long term trend is that the high values are staying constant or decreasing, then there is little likelihood that the limits will be exceeded. The rules check on the power system margin to determine if battery power is supplementing the solar cells during sunlight. These rules are still under development and are an area where much more can be added. As the system gets more fully defined and tested, more operational rules can be written to analyze telemetry.

The last set of rules write all conclusions to files so that the information may be presented to the user. The first files are the color files. All colors are assumed to be green

unless information in these files change their state to yellow or red. The first file is the "colorsys.dat" file which contains the color scheme for the top level categories of telemetry points. The file format is

```
temp red
power red
```

After analysis is complete, this information is read and applied to the telemetry color scheme for user display.

The second color file is "colors.dat", which contains the color scheme for each particular telemetry point that went out of range. The format for this file is

```
rxtemp 2 red
cell 12 yellow
batcur 2 red
```

This file is also read after analysis by the telemetry display routines, so that the user has a quick indication of problems in the telemetry information.

The last telemetry file is the "teldata.dat" file. This file contains all pertinent information about each data point as the system determined from the telemetry file. This includes the highest and lowest values found, the latest value, and the number of times that the telemetry point was outside of normal operating parameters. Each data point has its own significance. The highest and lowest values gives the user an idea of how close the telemetry point may be to its limits, or how far it exceeded them. The counter gives an idea as to how often the point was out of limits. If the number is very low, it may have been a single event upset, or some inconsequential event. A high count rate would alert the user that the system has exceeded the limits, and action needs to be taken. The order of the points is pre-determined, so a procedural display algorithm will be easy to implement.

There are also outputs that are appended to previous data that can aid the user in determining the current state, and trends. The first is "lifeval.dat" which is the lifetime values for each point. This file contains every reading of each telemetry point and associated time tag. This file is in a space delimited file format that can be easily

imported into a spread sheet or plotting routine for analysis. The next two files look at periodic characteristics. The rules determine the high and low value per orbit, and write those to "hivals.dat" and "lovals.dat". These two files strip away the transient readings and can aid in determining if the long term trends are getting closer to the limits, and if a prediction can be made as to when these points may exceed their operating limits. This will make specific trends much easier to notice. If certain components are getting hotter over a period of time, then the highest point per orbit will quickly point this out, while the constant temperature fluctuations that occur every orbit may obscure the long term trends.

The last file to be written are the conclusions that the program reached. The first is problems in the telemetry asserted in the limit checking and system operation routines. The "prob.dat" file contains all systems and readings that are not correct. The last file is the decisions file, "dec.dat" which contains all of the decisions reached by the expert system. A much more in depth written explanation of each rule and a rule interdependency matrix can be found in Appendix A. Appendix B contains the code developed so far in this project, which is not in its final form.

IX. GROUND STATION SOFTWARE DESIGN

A. REQUIREMENTS

The PANSAT Ground Station will be responsible for all tasks that will be associated with the daily operations of the satellite, as well as the primary contingency station in case unforeseen circumstances require special commanding from the ground. The ground station will be used by the Space Systems Academic Group, as well as students at the Naval Post Graduate School. The use of the system by untrained students requires that the user interface with the satellite be as simple to understand as possible without losing any of the capabilities required to effectively operate the satellite.

The ground station needs to have the capability to do several different operations in order to maintain the satellite system. First of all, the ground station will have to be able to track the satellite. This will require the station to have an accurate internal clock, and the ability to update it from an accurate source. The ground station also requires up to date ephemeris information on the satellite. Once the station knows the position of the satellite, and its predicted rise and set times, the station must then be able to control the antenna so that communications can be established.

Once the computer determines the satellite is in sight, then it must be able to establish a communications link. Since the satellite will be in a receive only mode, the ground computer will have to initiate the communications link. Once the link has been established, then the ground station must be able to transmit information, both commands and files, to the spacecraft, and receive information from the spacecraft. This information must also be displayed to the user in a logical manner, so that it can be quickly interpreted.

B. GROUND STATION DESIGN

The ground station will use one computer for controlling the antenna with a Kansas City Tracker. With one computer responsible for antenna pointing and satellite tracking, the user will be able to also display the satellite's current location, so the position of the antenna can be verified. The satellite's ephemeris will be updated via modem with data from the NPS amateur radio club bulletin board. The internal clock can also be updated by modem or radio to ensure accurate tracking.

A second computer has the satellite communications software as diagrammed in Figure 9-1. This program is menu driven to quicken and simplify use. The spacecraft operations are separated into three distinct phases. During the first phase the user sets up the ground station to prepare for the next satellite pass. The second phase is when the satellite is in view, and the ground station is communicating. The last phase is the post processing routines.

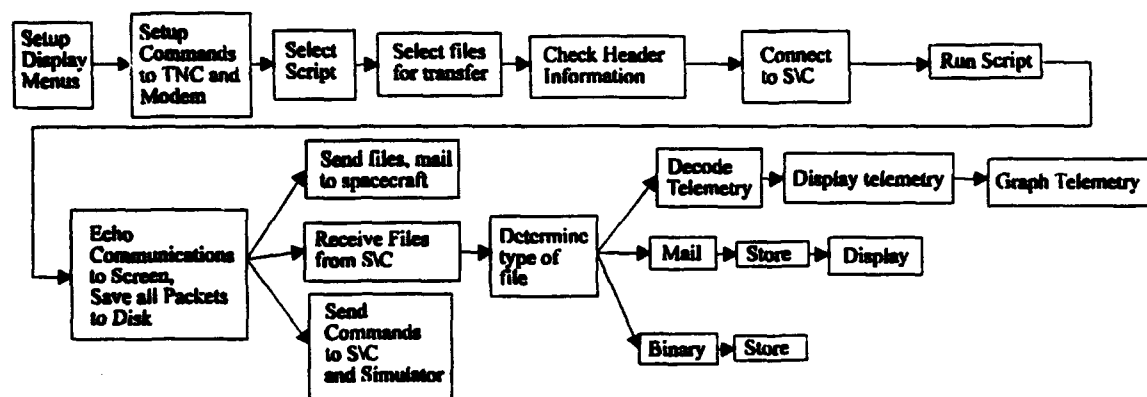


Figure 9-1. Top Level Ground Station Software Block Diagram

All of the commands performed in the pre-pass phase can also be performed during the satellite pass, but the limited time the satellite is in view discourages this use. Upon starting the ground software, it first sets up the display menus so that the user can begin operations. The modem and terminal node controller (TNC) are setup with configuration files. The user can edit a script for use on the next pass. It is written in a ASCII based text editor, and then compiled to ensure that all commands are understood and verified.

Files are also selected for transfer. The program automatically checks those selected for PANSAT file headers. If there is not one, the user is queried to supply the needed information. Once a script is selected, and the files are processed for transfer, the ground station is ready to connect to the spacecraft. A default script can also be used so not every pass will have to be monitored by ground personnel.

Once the ground station is in contact with the satellite, it will send a connect packet and establish a communication link. The ground station and satellite conclude the operations that were being performed the last time the satellite was in view. If the satellite had not yet completely downloaded a file, then the ground station will request to satellite to fill the holes in the partially downloaded files. Similarly, the ground station request and continues to upload any files that were only partially uploaded on the previous pass. Once these operations are completed, the script can be executed, or the user can send commands as he wants. For every pass, the ground station should at least perform all necessary functions to check on the 'health' of the satellite, and download or upload any messages.

Once the telemetry is downloaded, it is automatically sent to the CLIPS subroutines to determine the satellite state. The telemetry file is translated into format so the Artificial Intelligence routines can quickly access the information, search for conclusions from the data, and output its analysis to the user. Once the analysis is complete, the problems and decisions, as well as telemetry information can be accessed from the main menu, and displayed in a color coded windows format. The color coded telemetry categories will quickly cue the user to any problems in the satellite. A green indication signifies that the parameter is within its normal range for all operations. A yellow telemetry point indicates that an anomalous value that exceeded the normal operating range of the satellite, and could eventually impact the satellite's operation. A red indication means that a telemetry point has exceeded a critical threshold. The occurrence had endangered the spacecraft, and interferes with proper operation. Corrective action must be taken immediately, if not already performed by the autonomous nature of the satellite. The ground station will have

pull down menus to access the subcategories telemetry points as shown in Figure 9-2. Only the master ground station will have the ability to receive all telemetry files. Civilian

Temperature		Power		SCOS		BAX	
Scripts	Files	Super User Commands	Ground Commands	General Commands	A.I.		
<div>CMD:</div>							

Figure 9-2, Main Page Layout

users will only be able to download the current telemetry file. The satellite telemetry will be divided into categories and displayed as TEMPERATURE, POWER, SCOS, and BAX.

If a user wants amplifying information on a particular system, he will click the mouse on the major category, and the telemetry will be displayed in a window, with the description and color coded limits as illustrated in Figure 9-3. Any telemetry point can be selected for display by clicking the mouse on subsystem, which will bring up another

Temperature		Power	SCOS	BAX
cell 1	<div> <div>Temperature</div> <div>cell 10</div> <div>latest value 120</div> <div>worst value 125</div> <div>high 115</div> <div>normal high 100</div> <div>normal low 20</div> <div>low - 10</div> <div># times bad 6</div> <div>Graph telemetry point</div> </div>	Super User	General	A.I.
cell 2		Commands	Commands	
cell 3				
cell 4				
cell 5				
cell 6				
cell 7				
cell 8				
cell 9				
cell 10				
cell 11				
cell 12				
cell 13				
cell 14				
cell 15				
cell 16				
cell 17				
bat a				
bat b				
dcs a				
dcs b				
bus				

Figure 9-3, Telemetry Subsystem Display with Selected Telemetry Point

window that identifies the parameter, its latest value, high and low values, the limits, and the number of times the point has failed that criteria.

Another section of the main screen will display the raw communications that are going on with the satellite and will show the commands that are being sent and received by the satellite. The visual cue of the satellite link will also insure the proper operation of the automated software.

By selecting a menu option, the user will be able to send commands to the satellite. The commands are divided and accessed in three categories, Super User commands, general commands, and ground commands, shown in Figure 9-4. The super user commands are those that affect the configuration of the satellite. While general commands are other functions the ground station can have the satellite perform, but they do not alter the satellite. Neither of these categories are available to the civilian user. The last set of commands are the ground commands. These are the file transfer and mail commands that all civilian users will be able to use.

Temperature		Power	SCOS	BAX	
Scripts	Files	Super User Commands	Ground Commands	General Commands	A.I.
Select Edit Delete Compile	Add header Select to U/L Select to D/L Subdirectory for U/L Subdirectory for D/L	Super User Exit Super User Clear Cmds Clear Events Set Clock Ld Software Run Software Del Software Set Tel Batt Trickle Batt Charge Batt DCS Max TX Set Mode Set Atten	Dump List Dump Mail Post Bulletin Remove Bulletin Purge Mail Read Cur Tel Read Stor Tel Del Stor Tel Read OS Par Read Data Read Clock Read Cmds Read Events	List Read Del Send Logon Logoff Read Tel	Analyze file
CMD:					

Figure 9-4, Satellite Commanding Menus Layout

During the post pass phase, the data from the recent and past passes can be further analyzed, and stored. The data is also archived on a regular basis.

The other type of ground station will be designed for the civilian user. He can use either software offered by NPS, or may be able to use others commercially available that implement PACSAT FTL0 if the NPS version is compatible. These packages will not have the passwords needed to access the stored telemetry, or send commands to the spacecraft. It will allow the amateur full access to transfer files in the store and forward capability. The satellite will also have recent telemetry for those wishing to download that information. The spread spectrum transmitter will also be required. A kit will be available either through NPS, or a civilian company. With easily available software and hardware, the PANSAT should have numerous customers wanting to use its services.

C. FILE TRANSFER

When a file is to be transmitted to or from the satellite, the File Transfer Level 0 (FTL0) communication protocol, as seen in Figure 9-5, is used. It was developed by Jeff

Ward and Harold E. Price to be used on Pacsat. When a FTL0 packet is transmitted to the spacecraft, it has an information field of 0 to 2047 bytes, and a two byte header that indicates the number of bytes in the information field and the packet type. The type of information field uses binary 0 to 31, except for 18-29, to call subroutines onboard the satellite. The following information field can then be used by the satellite operating system.(Ref. 17)

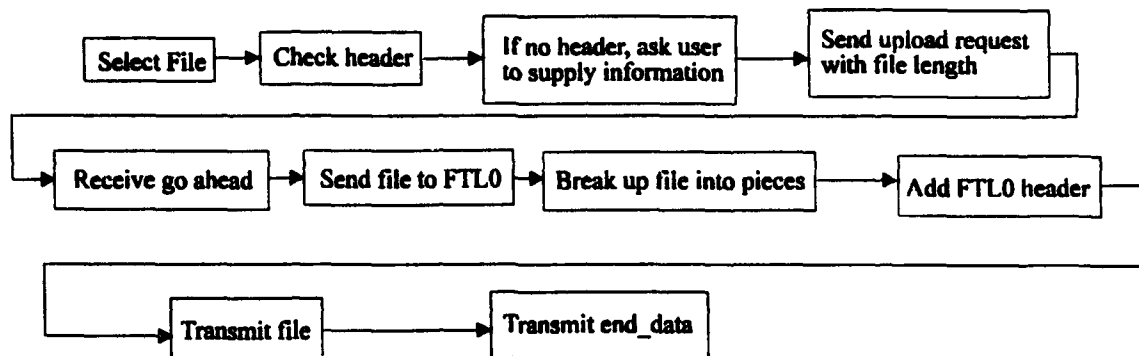


Figure 9-5, File Transfer Block Diagram

D. ADD PANSAT FILE HEADER

The file header added onto the front of every file transmitted to the spacecraft carries pertinent information for the satellite to properly carry out its functions. The file header consists of several fields, of which only a few are supplied by the user. When the user selects a file for upload, the ground station executes the file header routine on the file selected. The program prompts the user for file type and compression used. It then queries the user for the destination callsigns. The last step is for the user to supply the title of his message, and keywords so others can decide if they would like to download that file. Once the information is supplied, it is appended onto the front of the file along with other information added by the computer. The computer added fields include flag, length of file, source, priority, mail name and extension, number of destinations, title length, key words length, and checksums for the header and body.

When the ground station is linked to the satellite and requests to send a mail file, the computer first checks the PANSAT file header for validity. It then sends the upload request to the satellite along with the total size of the file. The satellite then determines if there is room in the mailbox for that size of a file. If so, it sends the ground station a number which is then used to identify the message. With this message number, the ground station can then upload the file. Once the message is received by the satellite, it puts additional information into the header. This includes body offset, download count, upload time, expire time, and updates the header checksum. When the ground station downloads a file, the computer will then strip the file of the PANSAT file header and save it on the hard disk.

E. TELEMETRY SUBSYSTEM

The telemetry system will have several different phases before it can be analyzed by the ground station user. The satellite digital control system will control multiplexers which are connected to A/D converters which read the various temperature, voltage, and current probes throughout the spacecraft. These values will be read into memory as a hexadecimal, unsigned value. The different sensors can be polled at different rates which can be set from the ground, depending upon the demands of the subsystems that require the information. The polling will also be conducted at regular intervals, and stored into a file. The format of the file will have the time that the sample was taken at, followed by the value of each telemetry point in order. This file will be binary and must be interpreted by the ground station. This file can then be transmitted to the ground station over the communications link just as any other file would be transmitted. The satellite also keeps in memory the latest telemetry points taken, and these too can be transmitted down to the user. The files will have a PANSAT file header added to the beginning before transmission, so that the ground station will know that the file contains telemetry information, and where the information is located. Once the ground station receives the telemetry file, each data point needs to be converted into a measured value according to

the equations that govern the sensors. The data is then used in the A.I. software so that vital statistics can be determined and presented to the user.

Once the ground station has converted the raw data into a usable format, the PANSAT telemetry analysis tool can then be used to print problems and solutions to the screen, or to a file. From the analysis, there can be several different types of recommendations. The first type is that the user can then command the spacecraft to reconfigure the satellite to operate in a mode that is unaffected by the subsystem that is giving the bad telemetry. Another possibility is that the operations of the satellite can be changed so that it is less affected by the telemetry points exceeding limits. The last possibilities are that the problem cannot be overcome, and the user can only monitor the situation, or the exceedance is the actual operational limits of the spacecraft, and the limits should be changed to accommodate.

X. SPACECRAFT COMMANDING

One most important aspect of the ground station will be the ability to send commands to the spacecraft that can affect the configuration and operations, as well as perform trivial housekeeping functions. For most subsystem functions, the software onboard the spacecraft must be able to autonomously operate the spacecraft. If situations arise where an action must be performed or risk the loss of the satellite, then the vehicle itself must be able to determine any action to be performed, and command itself. These events and decisions are stored in the spacecraft's event log, which can be downloaded and verified for proper operation.

A problem arises that not all situations can be predicted, so not all solutions can be known beforehand. To aid in this situation, the ground station must be able to command the spacecraft with all of the same commands that the spacecraft has as its own capabilities. The ability to command the spacecraft will also aid in troubleshooting the spacecraft and determine possible malfunctions in the system. The ground station will also be able to command the satellite to use different subsystems to insure that the redundant systems are still operational if desired.

There will be several different sets of commands that will be used. The first set will be those that can be used by all users; the NPS ground station, and the civilian users. These will be the basic commands to read mail, send mail, and download recent telemetry. These commands will be will comply with already existing FTL0 formats in wide use by amateur satellites. The only difference will be in the stations' program to put the proper PANSAT file header information onto the front of the file. Currently, the PACSAT file header protocol adds extra unneeded information that the PANSAT file header will omit. This primary reason is to shorten the overall file length.

The next sets of commands will be those that only the main ground station can execute. These will be broken into two categories. The first will be general housekeeping

commands. These will include the ability to purge mail, add or delete bulletins, download recent and stored telemetry, and other trivial tasks.

The last set of commands will be those that alter the configuration of the spacecraft, or command it to perform certain functions. These commands are known as Super User Commands, and will only be executed by authorized personnel. These commands include the ability to switch processors, DCS's, batteries, and load software. These commands can be used only if the correct password is used, and the computer will verify with the user that these commands are what the user wants to perform. These commands will also be executed in hexadecimal format. An advantage to this capability is that unauthorized users will be less likely to decode the commands that can manipulate the spacecraft in ways that we don't want them to be able.

There are several steps in sending a command to the spacecraft, diagrammed in Figure 10-1. When a command is selected, the computer verifies with the user that it is the proper command. The next step is to send the packet to the ground-based spacecraft simulator. This is an operational mockup of the satellite, which is in the same configuration as the ground station expects the satellite to be in. The ground simulator echoes back the command, then executes it. The ground station then verifies that the simulator echoed back the proper command, and executed it correctly. Once all of these steps have been successfully accomplished, then the command can be sent to the spacecraft.

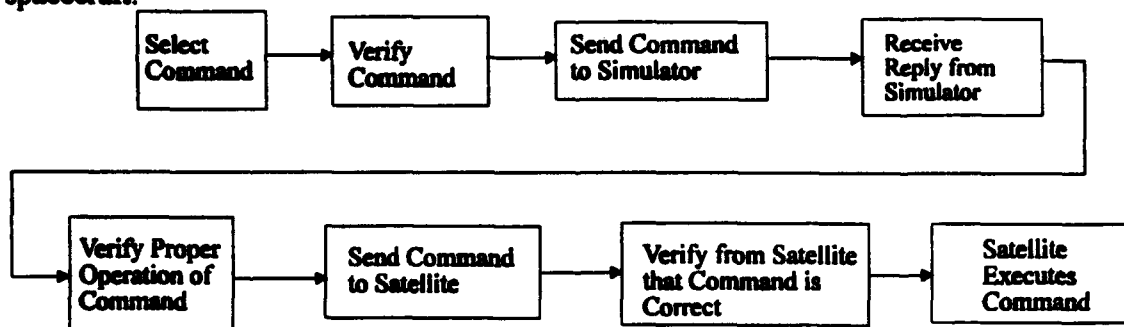


Figure 10-1, Spacecraft Commanding Block Diagram

In order to send a command to the satellite, it must be sent to a particular application on the spacecraft. The FTL0 protocol sends its information packets to PANSAT-1 where

-1 indicates the subsystem identification (SSID) which is application address on the satellite used by BAX. The commands to the satellite will be sent to PANSAT-2, which is the application for interpreting commands. The information sent to the command interpretation routine includes a one byte header that tells the satellite how large the packet is. Each command uses two bytes, the modifiers use one byte, and callsigns use 12 bytes. The largest size for the command set is 15 bytes, so only one byte is needed to count total number of bytes in the command. A packet for a ground control command like READ CUR TEL would then be 20B or in binary,

count	command
0002	00001011

Similarly, a packet for a subsystem command DISCH BATT \A would be 341 or in binary

count	command	modifier
0011	00110100	0001

If the satellite receives a subsystem command, it first checks that the user has already sent the SUPER command with proper password. If not, the satellite responds with an error message. If the command is valid, it transmits the interpreted command and waits for verification from the ground station. Once the satellite receives verification on the command that it interpreted, then it will execute the command, and transmit compliance. The list of commands is in Appendix C, while the binary formats are not included to keep the distribution of this thesis unlimited.

XI. CONCLUSIONS AND FURTHER STUDY

The automatic analysis of telemetry saves the ground station operator from the monotonous task of sorting through and analyzing the vast amounts of telemetry that are received from the satellite. By storing the project engineers' knowledge, the ability for inexperienced users to understand the health of the satellite is dramatically improved. PANSAT also has many students who work on the project and leave, and project engineers who will move onto other projects once the satellite becomes operational. The ability to retain the corporate knowledge is essential in providing continuity in the program.

From this point, there is still much work to be accomplished. The satellite is entering the testing phase, where the operational limits and procedures become more defined. As this happens, rules can be added to the knowledge base to keep the A.I. portion up to data with the actual configuration of the satellite. The coding for the ground station software can also begin as the interface between the controllers and satellite are tested. The A.I. code also needs to be imbedded in the C procedural code, where significant changes can occur. Instead of writing all decisions and problems to file, they can be presented to the user as soon as they are discovered through subroutines called by the rule set. As the program stands, it asks the user for the name of the telemetry file. In the final form, there are two different versions of the rule set. One will have the full capabilities, including the ability to append data to the lifetime and high and low values data files. This rule set will be used automatically when the ground station receives a new telemetry file from the spacecraft. The interface is automatic, and not seen by the user. Another version of the rule set is used to analyze past data sets. This rule set does not have the ability to append data to existing files, which would corrupt the data files by doubling old data out of sequence.

XII. LIST OF REFERENCES

- (1) Navy Satellite Operations Center, *ISCS Manual Status Indicator Program*.
- (2) Advisory Group for Aeronautical Research and Development, Conference Proceedings of Machine Intelligence for Aerospace Systems, AGARD Conference Proceedings 499, *Spacecraft Electrical Power System Fault Detection/Diagnosis and Resource Management*, by Peter Adamovits and Eric Jackson, pp. 1-5, 13-16 May 1991.
- (3) Advisory Group for Aeronautical Research and Development, Conference Proceedings of Machine Intelligence for Aerospace Systems, AGARD Conference Proceedings 499, *Advance Satellite Workstation (ASW)*, by T. Bleier, S. Hollander, and S. Sutton, pp 1-3, 13-16 May 1991.
- (4) Martin Marietta Corporation, *The Multi Purpose Causal (MPC) Tool User's Manual*, June 1991.
- (5) Proceedings of the First CLIPS Users Conference, NASA/Johnson Space Center, NASA Conference Publication 10049, *Interpretation of Space Shuttle Telemetry*, by Donald Culp, pp. 290-295, 13-15 August 1990.
- (6) Proceedings of the First CLIPS Users Conference, NASA/Johnson Space Center, NASA Conference Publication 10049, *A PC Based Fault Diagnosis Expert System*, by Christopher Marsh, pp. 442-443, 13-15 August 1990.
- (7) The Aerospace Corporation, Report SD-TR-87-56, *Expert System Prototype for False Event Discrimination*, by April Gillam, Paul Mazaika, and Pamela Feldman, pp. 5-10, 14 November 1985.
- (8) Naval Oceanographic and Atmospheric Research Laboratory, Technical Note 286, *Oceanographic Expert System: Potential for TESS(3) Applications*, by M. Lybanon, pp. 1-5, July 1992.
- (9) Chin-Hwa Lee, CS3310 Class Notes, April, 1993.
- (10) Advisory Group for Aeronautical Research and Development, Conference Proceedings of Machine Intelligence for Aerospace Systems, AGARD Conference Proceedings 499, *A Synergistic Approach to Reasoning for Autonomous Satellites*, by James M. Skinner, George F. Luger, pp 1-6, 13-16 May 1991.

- (11) Patrick Winston, *Artificial Intelligence*, Third Edition, Addison-Wesley Publishing Company, June 1992, p. 163.
- (12) Software Technology Branch, Johnson Space Center, JSC-25012, *CLIPS Reference Manual, Volume I, Basic Programming Guide*, 2 June 1993, pp. 26-30.
- (13) William B. Zeleny, Naval Postgraduate School, PH2511 Lecture Notes, January 1992.
- (14) Brij N. Agrawal, *Design of Geosynchronous Spacecraft*, Prentice-Hall, 1986, pp. 265-381.
- (15) Allan Kraus, Naval Postgraduate School, AE3804 Lecture Notes, April 1992.
- (16) Spectrolab, Inc., No. 0029387, *Power Analysis NPS*, by S. Missirian, 21 July 1993.
- (17) Jeff Ward and Harold E. Price, *Pacsat Protocol: File Transfer Level 0*, Date Unknown

APPENDIX A.

Explanation of rules for PANSAT telemetry analysis

The following paragraphs are quick explanations of each rule in the rule set. The number of each rule is used in following matrices of rule interdependence. The program can be run either in a windows environment, or DOS. After entering the operating environment, the user must type (load prog.clp) to load the constructs into memory. A (reset) must be commanded next, which clear the facts and asserts (initial-fact), which must exist for the program to execute. The program can then be executed with a (run) command. The program asks for a file name that contains the telemetry data, "data.clp" has data to test the program. Upon completion, all files are closed with a (close) command so that other applications can access those files.

1. defrule open-file

This operation is performed first because of the initial-fact which is automatically asserted with a (reset) command. The rule then asks the user for the file name that contains the telemetry data. In the final form, then telemetry file will be included, and the input will be transparent to the user. The file is then found and opened. The name of the file is saved for later use, and the initial-fact is removed so that the rule will not fire again. The rule also opens the output files, so that other rules may write or append data to them. Several facts are asserted to initiate the operation of several other rules.

2. defrule read_parameter

Here, the rule makes sure the file containing the current configuration file is open, and ready to be read which is asserted by either the open-file, or read_setting. It then reads the data of the configuration system and retracts the facts that fired the rule.

3. defrule read_setting

If the configuration file is open, and the subsystem name has been read by `read_parameter`, and it is not the end of file marker, then the rule reads in the subsystem selected, saves the information, and deletes the fact that fired the rule.

4. `defrule eof_setting`

If the configuration file is open, and the subsystem name has been read by `read_parameter`, and it is the end of file marker, then the rule closes the configuration file, and retracts the facts that fired the rule.

5. `defrule read-data-header`

The rule checks for the open data file name, and if the telemetry point name is to be read by a fact asserted by either `open-file` or `end-of-line`. The new telemetry point name is then read, as well as the next field that determines the number of those telemetry points. The telemetry name is then stored, along with the number of points. A counter is then initialized, and the fact that fired the rule is retracted.

6. `defrule read-data`

This is the only rule that has the salience set. It is so that the rule will only fire if it is the only rule on the agenda list. This is vitally important to the operation of the program. Since this rule deletes old telemetry points each time it reads in a new one, then is rules get under the `read-data` rule in the agenda list, they may no longer be valid after the rule deletes the old telemetry point. By setting the salience, this ensures that all rules associated with each telemetry point will fire before that point is deleted. If the points were not deleted, then the program would quickly fill up the available RAM, and the operation would slow down immensely as the data is stored in the hard disk.

The rule checks for the open data file name and an old data point. The next points name is found, and it is checked to make sure that it is not an end of file marker or time tag. The number of points is then found, and checked against the counter to make sure

that not all of the telemetry points for that subsystem have been read. The next telemetry point is then read and saved, and the old telemetry point is deleted. The point is also asserted as the latest point to begin the search for the latest value of each telemetry point. The rule also asserts facts to start the out_count rule in determining if an error is always happening. The counter is then incremented for the next read-data rule firing. The last two operations are to append the data points onto the life time value file, and assert a fact so that the limit checking only occurs once per telemetry point.

In a CLIPS program, the inference engine matches facts to rules, and only fires a rule once for a given set of facts that matches the left hand side. In the limit checking rules, a counter is incremented. Each time the rule fires, it deletes the old counter and asserts a new counter. This new counter is then a new fact that satisfies the left hand side. If another fact from a different routine were not asserted here, and retracted in the limit checking rules, then the limit rule would continue indefinitely, incrementing the counter continuously, which puts the program into an infinite loop.

7. defrule read-time

The program checks for open data file name, and finds the old number of telemetry points, counter, time fact, and time tags. It fires on assertions made in read-data-header. Then the new time is read, and the all old telemetry system information is deleted. It also asserts facts to save the time tag, and fire the read-data-header rule. The rule then prints a carriage return line feed to the life time values files to keep it in usable format.

8. defrule end-of-line

The current telemetry name is found from read-data-header, and checked to ensure the end of file has not been reached. The total number of points and counter is found and compared. If the counter is equal, then it has already read all of the data points, and the end of the telemetry data line has been reached. This causes the counter and number of

points and system read to be deleted, and the next telemetry data line can be read since read-data-header uses a fact asserted in this routine.

9. defrule end-of-file

The old total number of points, counter, and open data file name are found. The telemetry point from read-data-header is checked to see if the end of file marker has been read. If the program has reached the end of the data file, then the telemetry information and point, open data file fact and end of file point are deleted, and the data file is closed. A fact is asserted that the file is now closed to be used in other rules.

10. defrule start_counter

This rule initialized a counter to determine the number of times that a telemetry point is out of limits. A telemetry point is found, and if a counter does not exist yet that is associated with that telemetry point, then one is made. This counter is used in all of the limit checking rules.

11. defrule save_battery_current

This rule saves the battery current so that it is not deleted before it can be used with the solar cell current telemetry point. Otherwise, the read-data rule would delete the data point, and it could never be compared.

12. defrule save_cell_current

This rule is very similar to the save_battery_current since the two need to be used concurrently in other rules.

13. defrule in_sun "Check to see if the satellite is in the sun"

The solar cell current is found and checked to see if the cells are supplying significant power to the spacecraft bus. If it is, then the satellite is in the sun using primarily solar power.

14. defrule in_eclipse "Check to see if the satellite is in eclipse"

The battery current is found and checked to see if the battery is discharging. Then the same time tagged solar cell current is found to see if it is supplying significant power. If the power is coming from the batteries, then the satellite is in eclipse and that fact is asserted and time tagged.

15. defrule exceed_power_budget

In this case, the battery current is found, and tested to see if it is supplying significant power. A solar cell current is also found and tested if it too is supplying significant power. If both of the power supplies are giving power to the spacecraft, then the satellite must be in the sun, as has been found using `in_sun`, but also, the requirements of the spacecraft exceed the ability of the solar panels to supply all needed power. This means that the spacecraft is operating in a negative power budget. This rule recommends to fully check out the power system, and consider changing spacecraft operational capabilities when it finds that the power budget has been exceeded.

16. defrule delete_old_power_facts

When the read-data rule is completed, and the data file is closed, no more telemetry points will be read. This rule deletes all of the battery temperature and current points as well as the solar cell currents that were saved, since they will no longer be needed.

17. defrule save_battery_temp

This rule is very similar to the `save_battery_current` since the battery temperature and voltage need to be used concurrently in other rules.

18. defrule save_battery_volt

This rule is the same as to the `save_battery_temp` since the battery temperature and voltage need to be used concurrently in other rules.

19. defrule batt_charged

This rule determines if the batteries are fully charged, and the time that they are first fully charged in each orbit. By linearizing the suppliers graph voltage and temperature, the equation of a line will determine if the batteries are at or above the voltage threshold for a given temperature. If so, the rule determines that the batteries are fully charged.

20. defrule bat_temp_increase

When a battery is fully charged, continued charging causes the temperature of the battery to increase, since the energy is not used for increasing the electric potential of the battery. This rule determines how much the temperature rises after it has been decided that the batteries are fully charged. The fact that the batteries are charged is found, and the battery temperature with the same time tag, so the temperature at completed charging is known. Another battery temperature fact is found. The facts are then tested to ensure that the second temperature occurred within one orbit after the first reading, and the second point is higher than the first temperature. The rule then finds the temperature change and saves it as a new fact.

21. defrule largest_change

Since there are several readings after the batteries are charged, there will be several temperature differences. This rule finds two temperature difference facts in the same orbit and retracts the smaller difference. Eventually, only the largest temperature increase will remain.

22. defrule latest_eclipse

Here, the rule finds two times in eclipse, and determines which one is later. It then deletes the earlier time in eclipse.

23. defrule latest_sun "Find latest time in sun for current orbit"

The computer finds two times that the satellite is in the sun, and determines which is later. It then deletes the earlier time in the sun.

24. defrule leave_eclipse

If there is an in eclipse fact followed by a later (in_sun time) fact, then the satellite just left the earth's shadow. In this case, the rule deletes the (in eclipse time) fact, and stores the time that the satellite left the earth's shadow as a new fact.

25. defrule find_latest

If this rule finds two latest facts as initiated by the read-data rule, it will delete the earlier fact. Eventually, only one fact remains, and is output in the "teldata.dat" file once all data has been read.

26. defrule solar_cells_cold

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a solar cell temperature point is searched for and checked to see if it is below the lower temperature range. The counter for the cell is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The red telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

27. defrule solar_cells_cool

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a solar cell temperature point is searched for and checked to see if it is above the lower temperature range, and below the lower normal operating temperature. The counter for the cell is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The yellow telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

28. defrule solar_cells_warm

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a solar cell temperature point is searched for and checked to see if it is above upper normal operating temperature, and below the upper temperature range. The counter for the cell is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The yellow telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

29. defrule solar_cells_hot

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a solar cell temperature point is searched for and checked to see if it is above the upper temperature range. The counter for the cell is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The red telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

30. defrule bat_volt_too_low

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a battery voltage point is searched for and checked to see if it is below lower voltage range. The counter for the battery voltage is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The red telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented. This rule determines that whenever the battery voltage gets too low, the power system should be checked out, and recommends either reconditioning the batteries, or changing operations so that the batteries do not drain as much during eclipse.

31. defrule bat_volt_low

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a battery voltage point is searched for and checked to see if it is above lower voltage range, and below the lower normal operating voltage. The counter for the battery voltage is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The yellow telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented. The same recommendations are made as for bat_volt_too_low.

32. defrule bat_volt_high

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a battery voltage point is searched for and checked to see if it is above upper normal operating voltage, and below the upper voltage range. The counter for the battery voltage is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The yellow telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented. If the battery voltage gets too high, the rule recommends trickle charging the batteries, which limits the charging current.

33. defrule bat_volt_too_high

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a battery voltage point is searched for and checked to see if it is above upper maximum voltage range. The counter for the battery voltage is also found. If it is out of limits, then the rule saves the telemetry point, voltage, and time tag. The red telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter

is incremented. If the battery voltage gets too high, the rule recommends trickle charging the batteries, which limits the charging current.

34. defrule bat_temp_cold

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a battery temperature point is searched for and checked to see if it is below the lower temperature range. The counter for the battery temperature is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The red telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

35. defrule bat_temp_cool

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a battery temperature point is searched for and checked to see if it is above the lower temperature range, and below the lower normal temperature operating range. The counter for the battery temperature is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The yellow telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

36. defrule bat_temp_warm

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a battery temperature point is searched for and checked to see if it is above upper normal temperature range, and below the upper temperature range. The counter for the battery temperature is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The yellow telemetry colors for this condition are saved,

the (new fact) is deleted, and the out of range counter is incremented. The rule recommends trickle charging the batteries at a reduced current flow.

37. defrule bat_temp_hot

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a battery temperature point is searched for and checked to see if it is above the upper temperature range. The counter for the battery temperature is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The red telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented. The rule recommends trickle charging the batteries at a reduced current flow.

38. defrule battery_current_high

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a battery current point is searched for and checked to see if it is above the upper normal current range and below the maximum current range. The counter for the battery current is also found. If it is out of limits, then the rule saves the telemetry point, current, and time tag. The yellow telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

39. defrule battery_current_too_high

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a battery current point is searched for and checked to see if it is above the upper maximum current range. The counter for the battery current is also found. If it is out of limits, then the rule saves the telemetry point, current, and time tag. The red telemetry

colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

40. defrule tx_current_high

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a transmitter current point is searched for and checked to see if it is above the upper normal current range and below the maximum current range. The counter for the transmitter current is also found. If it is out of limits, then the rule saves the telemetry point, current, and time tag. The yellow telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented

41. defrule tx_current_too_high

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a transmitter current point is searched for and checked to see if it is above the maximum current range. The counter for the transmitter current is also found. If it is out of limits, then the rule saves the telemetry point, current, and time tag. The red telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented

42. defrule bus_volt_too_low

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a bus voltage point is searched for and checked to see if it is below the minimum voltage range. The counter for the bus voltage is also found. If it is out of limits, then the rule saves the telemetry point, voltage, and time tag. The red telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented

43. defrule bus_volt_low

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a bus voltage point is searched for and checked to see if it is above the minimum voltage range and below the normal operating voltage. The counter for the bus voltage is also found. If it is out of limits, then the rule saves the telemetry point, voltage, and time tag. The yellow telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented

44. defrule bus_volt_high

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a bus voltage point is searched for and checked to see if it is above the normal operating voltage and below the maximum voltage range. The counter for the bus voltage is also found. If it is out of limits, then the rule saves the telemetry point, voltage, and time tag. The yellow telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented. The bus voltage is directly tied to the battery charging rate, so if the bus voltage is out of limits, this rule recommends to trickle charge the battery to limit the charging current flow.

45. defrule bus_volt_too_high

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a bus voltage point is searched for and checked to see if it is above the maximum voltage range. The counter for the bus voltage is also found. If it is out of limits, then the rule saves the telemetry point, voltage, and time tag. The red telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

The bus voltage is directly tied to the battery charging rate, so if the bus voltage is out of limits, this rule recommends to trickle charge the battery to limit the charging current flow.

46. defrule bus_temp_cold

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a bus temperature point is searched for and checked to see if it is below lower temperature range. The counter for the bus temperature is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The red telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

47. defrule bus_temp_cool

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a bus temperature point is searched for and checked to see if it is above lower temperature range, and below the lower normal operating temperature. The counter for the bus temperature is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The yellow telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

48. defrule bus_temp_warm

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a bus temperature point is searched for and checked to see if it is above the upper normal operating temperature, and below the upper temperature range. The counter for the bus temperature is also found. If it is out of limits, then the rule saves the telemetry

point, temperature, and time tag. The yellow telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

49. defrule bus_temp_hot

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a bus temperature point is searched for and checked to see if it is above the upper temperature range. The counter for the bus temperature is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The red telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

50. defrule tx_volt_too_low

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a transmitter voltage point is searched for and checked to see if it is below the minimum voltage range. The counter for the transmitter voltage is also found. If it is out of limits, then the rule saves the telemetry point, voltage, and time tag. The red telemetry color changes for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented. A high transmitter bus voltage is directly tied to the electrical power system bus, so there is most likely a problem in the over power system. The rule recommends checking the power system for voltages out of range.

51. defrule tx_volt_low

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a transmitter voltage point is searched for and checked to see if it is below the normal voltage range and above the minimum voltage range. The counter for the transmitter

voltage is also found. If it is out of limits, then the rule saves the telemetry point, voltage, and time tag. The telemetry color changes for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

52. defrule tx_volt_high

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a transmitter voltage point is searched for and checked to see if it is above the normal operating voltage and below the maximum voltage range. The counter for the transmitter voltage is also found. If it is out of limits, then the rule saves the telemetry point, voltage, and time tag. The yellow telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

53. defrule tx_volt_too_high

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a transmitter voltage point is searched for and checked to see if it is above the maximum voltage range. The counter for the transmitter voltage is also found. If it is out of limits, then the rule saves the telemetry point, voltage, and time tag. The red telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

54. defrule tx_temp_cold

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a transmitter temperature point is searched for and checked to see if it is below the minimum temperature range. The counter for the transmitter temperature is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The

red telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented. The recommendation is made whenever the transmitter temperature is low to transmit more often if the power budget allows.

55. defrule tx_temp_cool

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a transmitter temperature point is searched for and checked to see if it is below the normal operating voltage range and above the minimum temperature range. The counter for the transmitter temperature is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The yellow telemetry color changes for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

56. defrule tx_temp_warm

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a transmitter temperature point is searched for and checked to see if it is above the normal operating temperature range and below the maximum temperature range. The counter for the transmitter temperature is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The yellow telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented. A recommendation is made for a transmitter that is higher the temperature range may be able to be fixed by increasing the attenuation on the transmitted signal, so the transmitter will use less power, or switch transmitters.

57. defrule tx_temp_hot

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously,

then a transmitter temperature point is searched for and checked to see if it is above the maximum temperature range. The counter for the transmitter temperature is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The red telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

58. defrule rx_temp_cold

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a receiver temperature point is searched for and checked to see if it is below the minimum temperature range. The counter for the receiver temperature is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The red telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented. Any time a receiver is below normal operating temperatures, a recommendation is made to switch receivers and determine if the secondary has any problems.

59. defrule rx_temp_cool

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a receiver temperature point is searched for and checked to see if it is above the minimum temperature range and below the normal operating temperature. The counter for the receiver temperature is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The yellow telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

60. defrule rx_temp_warm

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a receiver temperature point is searched for and checked to see if it is above upper normal operating temperature, and below the upper temperature range. The counter for the receiver temperature is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The yellow telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

61. defrule rx_temp_hot

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a receiver temperature point is searched for and checked to see if it is above the upper temperature range. The counter for the receiver temperature is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The red telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

62. defrule dcs_temp_cold

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a digital control system temperature point is searched for and checked to see if it is below the minimum temperature range. The counter for the DCS temperature is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The red telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented. Whenever the DCS is outside of normal temperatures, the rules recommend switching DCSs.

63. defrule dcs_temp_cool

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a digital control system temperature point is searched for and checked to see if it is above lower temperature range, and below the lower normal operating temperature. The counter for the DCS temperature is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The yellow telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

64. defrule dcs_temp_warm

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a digital control system temperature point is searched for and checked to see if it is above upper normal operating temperature, and below the upper temperature range. The counter for the DCS temperature is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The yellow telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

65. defrule dcs_temp_hot

This rule first determines if a telemetry point has been limit checked earlier by finding the (new fact) asserted by read-data. If a point has not been checked previously, then a digital control system temperature point is searched for and checked to see if it is above the upper temperature range. The counter for the DCS temperature is also found. If it is out of limits, then the rule saves the telemetry point, temperature, and time tag. The red telemetry colors for this condition are saved, the (new fact) is deleted, and the out of range counter is incremented.

66. defrule edacseu_counter

The first step in this rule is to find the error detection and correction counter and test to determine if too many errors have occurred in the software. EDAC errors are most likely caused by radiation or particles striking the internal components of the spacecraft. With the error detection and correction, it can detect and correct one error, and detect two errors without correction. If the number of errors is large, then the probability of getting two errors which are not correctable in the same space greatly increases. The RAM wash is a subroutine to read all data in memory, do the error checking and correcting, counting all errors, and writes the data back into memory. This rule recommends increasing the frequency of RAM wash to lower the number of single event upsets in the RAM.

67. defrule worst_case_system

The main telemetry display will only show the color for the worst case of the categories subsystems. In every limit checking rule, if the same fact is asserted twice for the same system, CLIPS will only keep one assertion. Then only one red and one yellow fact at most will remain. In this case, the rule finds two facts for the same set of telemetry points, one red and one yellow. It then deletes the yellow case and saves the worst case, the red one. If there is only one yellow case, then it will remain as the worst case.

68. defrule worst_case_tel_pt

This rule is very similar to the worst_case_system, but instead of saving the worst case for the system, it is only saving the worst case for each telemetry point. Since at different times any telemetry point could be red or yellow, only the red case will remain.

69. defrule initialize_occurrence_problem

This rule is to start the process of finding when during an orbit a telemetry point goes out of limits. If a telemetry point goes out of limits every orbit, this algorithm will give the user an idea if it occurs at about the same time every orbit, or if it is not related in

any way to the orbital period. The rule finds a telemetry problem and the last time the satellite left eclipse and makes a fact of first time the problem occurred.

70. defrule find_first_occurrence

In this situation, the rule finds two different facts that state the first time a problem occurred, and determines which actually occurred first. within the same orbit, and deletes the latest occurrence.

71. defrule initialize_high_and_low_orbit

In the same manner as the find_first_occurrence rules, this set of rules finds the highest value per orbit of each telemetry point. The rule first finds a telemetry point that is an analog value, not a binary state or counter, and initialized that fact as the high for the orbit. This rule also asserts the fact as the highest value to start the search for the highest telemetry point in the entire file.

72. defrule find_high_orbit

The next step in the process is to find two high values of the same telemetry point that occurred within the same orbit, and delete the lower value. The high value is then written to a file at the end of each orbit with the save_hi_and_low_data rule.

73. defrule find_highest

Like the find_high_orbit routine, this rule searches for the highest value of each telemetry point in the file for presentation to the user. The highest value will give the user a quick idea of how close the telemetry point is to its limits. The highest facts have already been asserted, so this rule finds two highest facts, and deletes the data point with the lower value.

74. defrule find_low_orbit

Just like the find_high_orbit, this rule finds two low values of the same telemetry point that occurred within the same orbit, and deletes the higher value.

75. defrule find_lowest

Like the find_low_orbit routine, this rule searches for the lowest value of each telemetry point in the file for presentation to the user. The lowest value will give the user a quick idea of how close the telemetry point is to its limits. The lowest facts have already been asserted, so this rule finds two lowest facts, and deletes the data point with the higher value.

76. defrule check_configuration

This rule uses the configuration file that contains the information about the satellite's settings. This rule compares the ground's settings with the satellites to determine if the satellite has autonomously changed its configuration. Two facts are found, the configuration fact and the telemetry fact for the same system. The settings for that system are then compared. If they are not the same, then the fact that the satellite has changed its configuration is asserted.

77. defrule del_config

Once the telemetry data file is closed, there are no more settings to be compared, so the configuration facts are deleted.

78. defrule del_high

Once the telemetry data file is closed, the high data points per orbit are still in memory. Since the orbit will not be completed in the telemetry data, these are of no use since it is not known if these actually are the high values for the entire orbit. Therefore, this rule deletes all unwanted high facts.

79. defrule del_low

Once the telemetry data file is closed, the low data points per orbit are also still in memory. Since the orbit will not be completed in the telemetry data, these are of no use

since it is not known of these actually are the low values for the entire orbit. Therefore, this rule deletes all unwanted low facts.

80. defrule constant_out_counter

Once a telemetry point has exceeded a limit, it may either stay out of limits, or fall back within the limits. This rule will determine if the telemetry point stayed out of limits once it exceeded those limits.

This accomplished by initializing a counter that fires once an orbit after a telemetry point has exceeded its limits. If there is a limit exceedance and a counter hasn't been started yet, then a counter is started.

81. defrule out_count

This rule continues the constant_out_counter. The fact is fired by the data point that is being read by read-data. If a data point is read that has a counter already, then the counter is incremented and the fact that fired the rule is deleted.

82. defrule check_counts

Once the telemetry data file is closed, this rule checks on the counters to determine if the telemetry point stayed out of limits once it exceeded them. It gets two counters, the count that incremented each time the data point was read after it went bad, and the counter of the number of times that the telemetry point was out of limits. If these two counters are the same, then the telemetry point stayed out of limits once it exceeded the limits the first time. The counter since the first occurrence is deleted so that save_telemetry_data can fire. This prevents save_telemetry_data from deleting the counters before this rule can use them.

83. defrule not_always_out

This rule does the same process as out_count, except that it sees if the counters are different. If they are different, then the telemetry point did not stay out of parameters, and

the counter since the first occurrence is deleted. This allows the `save_telemetry_data` rule to fire.

84. defrule del_next

The read-data rule asserts rules so that the `out_count` routine can work. Once the telemetry data file is closed, these facts are no longer needed. This routine determines that the file is closed, and deletes the facts asserted by read-data

85. defrule save_first_problems

The problems with the spacecraft are written to an ASCII file so that they may be presented to the user in the graphical interface. When the telemetry data file is closed, a problem fact is found that was also the first time that the problem occurred. The two facts are then combined to inform the user that a problem occurred, the time tag, and the time into the orbit the problem occurred. The information is written to the "problem.dat" file and deleted from memory.

86. defrule save_problems

Problems that were not the first in its orbit are also saved to the problem file. In this case, the rule finds a fact, and determines that there is not a corresponding first time fact. The information is then written to the "problem.dat" file and deleted from memory.

87. defrule save_decisions

The decisions that the program arrives at are written to an ASCII file for display to the user once the telemetry data file has been completely read. If the telemetry data file is closed, then the rule writes all decisions that are recommended by the program to the "decide.dat" file. The decisions are then deleted from memory.

88. defrule save_telemetry_colors

Once the telemetry file is closed, the telemetry point colors are written to the ASCII file "colors.dat". This information is used to change the colors displayed to the user. If a

telemetry point is not specified by the file to be red or yellow, then the user interface assumes that the telemetry point is in the green. Once the color facts are written to file, they are removed from memory.

89. defrule save_system_colors

Once the telemetry file is closed, the telemetry system colors are written to the ASCII file "colorsys.dat". This information is used to change the colors of the major categories of telemetry displayed to the user. If a telemetry system is not specified by the file to be red or yellow, then the user interface assumes that the system is in the green. Once the color facts are written to file, they are removed from memory.

90. defrule save_hi_and_low_data

Here, the high and low data points per orbit are written to the "hivals.dat" and "lovals.dat" files. When there are two "sun" with times tags facts, then the satellite has completed an orbit. The rule then deletes the earlier "sun" fact to start the process again. The high and low values for each data point are then found and written to file. The first value in each line of the file is the time tag for the beginning of the orbit. Since each high and low value will occur at a different time for each telemetry point, only the orbit time is kept so that only two files need be used. The time tag is then followed by the high or low value for that orbit. The facts are written to the files in the same order as the telemetry stream.

91. defrule save_telemetry_data

The telemetry data is saved to a file "teldata.dat" so that the information can be displayed to the user through the telemetry display interface. This rule fires after the telemetry data file is closed, and the count_out rule no longer needs the data since it is deleted in this step. The rule finds all of the facts concerning each telemetry points highest, lowest, and latest values. It also collected each points' counter for the number of

times that it was out of limits. It then writes all of the information to file in an orderly fashion. The teldata.dat file will be in the same order as the telemetry stream. The first row will be the first telemetry point read in the file. The first number is the highest value that the telemetry point had in the file. The next is the lowest value. The third number is the counter of the number of times that the point was out of limits. The last number in each line is the latest value of that telemetry point. All of the data written to file is then retracted from memory.

The following matrices show the dependence of the rules on each other. Across the top of the matrix represents that left hand side of the rule, while down the left side represents the right hand side of the equation. The X's mark the assertions from the right hand side of the rules that cause the left hand sides to become satisfied.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1		X	X	X	X	X	X	X	X	X																					
2			X	X																											
3		X		X																											
4																															
5						X	X	X	X	X																					
6						X	X	X	X	X	X	X					X	X							X	X	X	X	X	X	X
7					X	X	X																								
8					X																										
9																X															
10																										X	X	X	X	X	X
11													X	X	X	X															
12												X	X	X	X																
13																								X	X						
14																							X		X						
15																															
16																															
17																X			X	X											
18																X			X												
19																					X										
69																				X											

	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61
1																														
2																														
3																														
4																														
5																														
6	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
7																														
8																														
9																														
10	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
11																														

	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46
26	X																				
27		X																			
28			X																		
29				X																	
30					X																
31						X															
32							X														
33								X													
34									X												
35										X											
36											X										
37												X									
38													X								
39														X							
40															X						
41																X					
42																	X				
43																		X			
44																			X		
45																				X	
46																					X

	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61
47	X														
48		X													
49			X												
50				X											
51					X										
52						X									
53							X								
54								X							
55									X						
56										X					
57											X				
58												X			
59													X		
60														X	
61															X

	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91
1																														
2																														
3															X	X														
4																														
5																														
6	X	X	X	X	X				X					X					X		X							X	X	
7																														
8																														
9															X	X	X			X	X	X	X	X	X	X	X	X	X	X
10	X	X	X	X	X															X	X	X	X	X	X	X	X	X	X	X
11																				X	X									X
12																														
13																														
14																														
15							X											X				X	X	X						
16																														
17																														
18																														
19																														
20																														
21																														
22																														
23																														
24							X	X	X																					
25																														
26					X	X	X											X				X	X							
27					X	X	X											X				X	X							
28					X	X	X											X				X	X							
29					X	X	X											X				X	X							
30					X	X	X											X				X	X	X						
31					X	X	X											X				X	X							
32					X	X	X											X				X	X	X						
33					X	X	X											X				X	X							
34					X	X	X											X				X	X							
35					X	X	X											X				X	X							
36					X	X	X											X				X	X							
37					X	X	X											X				X	X	X						
38					X	X	X											X				X	X							
39					X	X	X											X				X	X							
40					X	X	X											X				X	X							
41					X	X	X											X				X	X							
42					X	X	X											X				X	X							

	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91
43						X	X	X											X					X	X					
44						X	X	X											X					X	X	X				
45						X	X	X											X					X	X					
46						X	X	X											X					X	X					
47						X	X	X											X					X	X					
48						X	X	X											X					X	X					
49						X	X	X											X					X	X					
50						X	X	X											X					X	X	X				
51						X	X	X											X					X	X					
52						X	X	X											X					X	X					
53						X	X	X											X					X	X					
54						X	X	X											X					X	X	X				
55						X	X	X											X					X	X					
56						X	X	X											X					X	X					
57						X	X	X											X					X	X	X				
58						X	X	X											X					X	X					
59						X	X	X											X					X	X					
60						X	X	X											X					X	X					
61						X	X	X											X					X	X	X				
62	X					X	X	X											X					X	X					
63		X				X	X	X											X					X	X					
64			X			X	X	X											X					X	X					
65				X		X	X	X											X					X	X	X				
66																										X				
67																														
68																														
69									X															X	X					
70																														
71										X	X	X	X				X	X										X	X	
72																														
73																														
74																														
75																														
76									X										X					X	X					
77																														
78																														
79																														
80																				X										
81																				X										
82																													X	

APPENDIX B.

Code Developed for PANSAT Telemetry Analysis

```
(defrule open-file "Open the telemetry and configuration file"
  (initial-fact)                                     ;Do this operation first
=>
  (retract 0)                                         ;Prevent firing again
  (printout t "Name of file to read?" crlf) ;Ask file to read
  (bind ?name (read))                                ;Get file name
  (open ?name data "r")                              ;Open the data file
  (open "setting.dat" setting "r")                   ;Open the configuration file
  (open "lifevals.dat" lifeval "a")                  ;Open lifetime data file
  (open "hivals.dat" hivals "a")                     ;Open high value/orbit file
  (open "lovals.dat" lovals "a")                     ;Open low value/orbit file
  (open "problems.dat" problems "w")                 ;Open problem file
  (open "decide.dat" decide "w")                     ;Open decision file
  (open "colors.dat" colors "w")                     ;Open color file
  (open "colorsys.dat" colorsys "w")                 ;Open color file
  (open "teldata.dat" teldata "w")                   ;Open telemetry data file
  (assert (read-file ?name))                          ;Save name of file
  (assert (nextline))                                ;Set up to read file
  (assert (time 0))
  (assert (tel not 1 1 1))
  (assert (setting open))
  (assert (nextval))
)
(defrule read_parameter "Read the subsystem name"
  (setting open)                                     ;Ensure config file open
  ?next<-(nextval)                                  ;Get address of nextval
=>
  (bind ?set (read setting))                          ;Read subsystem name
  (assert (set ?set))                                ;Assert name
  (retract ?next)                                    ;Remove nextval
)
(defrule read_setting "Read the subsystem selected"
  (setting open)                                     ;Ensure config file open
  ?setting<-(set ?set)                               ;Get address of set fact
  (set "EOF")                                         ;Check not EOF marker
=>
  (bind ?val (read setting))                          ;Read the subsystem selected
  (assert (config ?set ?val))                        ;Assert subsystem selected
  (assert (nextval))                                  ;Read_parameter fires again
  (retract ?setting)                                 ;Retract old subsystem name
)
(defrule eof_setting "Read the EOF marker"
  ?set<-(setting open)                              ;Ensure config file open
  ?end<-(set EOF)                                    ;Find EOF fact
```

```

    ?next<-(nextval)                ;Find nextval fact
=>
    (close setting)                  ;Close configuration file
    (retract ?next)                  ;Delete nextval fact
    (retract ?end)                   ;Delete EOF fact
    (retract ?set)
)
(defrule read-data-header "Read the name of telemetry point"
  (read-file ?name)                 ;Check for open file
  ?next <- (nextline)               ;Get address of fact
=>
  (bind ?var (read data))            ;Read telemetry point name
  (bind ?num (read data))            ;Read number of points
  (assert(pt ?var))                  ;Store the point name
  (assert(tot ?num))                 ;Store the number of points
  (assert (count 1))                 ;Start counter
  (retract ?next)                   ;Retract to end
)
(defrule read-data "Read the data values"
  (declare (salience -100))          ;Allows all rules
  (read-file ?name)                  ;Check for open file
  ?olddata<-(tel ?varold ?valold ?dataold ?timold);del old data
  (pt ?var)                          ;Get point name
  (pt ~EOF)                          ;Check to see not EOF
  (pt ~time)                         ;Check to be not time tag
  (tot ?num)                         ;Get number of points
  (time ?tim)                        ;Get time tag
  ?countval <- (count ?val)          ;Get counter
  (test (<= ?val ?num))              ;Check counter
=>
  (bind ?data (read data))            ;Read to data point
  (retract ?olddata)                 ;Delete old data point
  (assert (tel ?var ?val ?data ?tim));Save new data
  (assert (latest ?var ?val ?data ?tim));Start latest search
  (assert (next ?var ?val))          ;Start constant out routine
  (bind ?val (+ ?val 1))             ;Increment counter
  (retract ?countval)               ;Delete old counter
  (assert (count ?val))              ;Save new counter
  (printout lifeval " " ?data)       ;Write to lifetime data file
  (assert (new fact))                ;Stop firing on limit checks
)
(defrule read-time "Read the time tag from the file"
  (read-file ?name)                  ;Check for open file
  ?oldtot <- (tot ?num)              ;Get old number of points
  ?oldcount <- (count ?val)          ;Get old counter
  ?oldtime <- (time ?tim)            ;Get old time tag
  ?pttime <-(pt time)                ;Get time fact
=>
  (bind ?t (read data))              ;Read new time tag
  (retract ?oldtime)                 ;Delete old time tag
  (retract ?pttime)                  ;Delete point
  (retract ?oldtot)                  ;Delete old total of points

```

```

(retract ?oldcount)           ;Delete old counter
(assert (nextline))           ;Prepare for next line
(assert (time ?t))             ;Save time tag
(printout lifeval crlf ?t)     ;Write time to lifetime file
)
(defrule end-of-line "Read to the end of line"
  ?pt <- (pt ?var)             ;Get point name
  (pt "EOF")                   ;Check not end of file
  ?total <- (tot ?num)          ;Get old total of points
  ?countval <- (count ?val)     ;Get old counter
  (test (> ?val ?num))         ;Check counter more points
=>
  (retract ?countval)          ;Delete old counter
  (retract ?total)            ;Delete old number of points
  (retract ?pt)               ;Delete old telemetry name
  (assert (nextline))         ;Prepare to read next line
)
(defrule end-of-file "Have reached end of file marker"
  ?tot <- (tot ?num)           ;Get old total of points
  ?count <- (count ?val)       ;Get old counter
  ?readfile <- (read-file ?name) ;Get data file name
  ?pteof <- (pt EOF)           ;Get end of file
  ?olddata <- (tel ?var2 ?num2 ?data2 ?time2) ;Get old telemetry
=>
  (retract ?readfile)         ;Delete file open fact
  (retract ?olddata)          ;Delete old telemetry point
  (retract ?pteof)            ;Delete EOF data point
  (retract ?tot)              ;Delete old number of points
  (retract ?count)            ;Delete old counter
  (assert (done-reading))      ;Store finished reading file
  (close data)                ;Close data file
)
(defrule start_counter "Start a counter for number of problems"
  (tel ?var ?num ?data ?time)  ;Get telemetry point
  (not (counter ?var ?num ?count)) ;Check no counter
=>
  (assert (counter ?var ?num 0)) ;Start counter
)
(defrule save_battery_current
  (tel batcur ?val ?amp ?min)   ;Get battery current
=>
  (assert (power batcur ?val ?amp ?min));Save in different form
)
(defrule save_cell_current
  (tel cellcur ?val ?amp ?min)  ;Get cell current
=>
  (assert (power cellcur ?val ?amp ?min));Save in different form
)
(defrule in_sun "Check to see if the satellite is in the sun"
  (power cellcur ?val ?amp ?min) ;Get battery current
  (test (> ?amp .01))           ;Check to see if cells lit
=>

```

```

    (assert (in_sun ?min))                ;Satellite in sun
  )
  (defrule in_eclipse "Check to see if the satellite is in eclipse"
    (power batcur ?val ?amp ?min)         ;Get battery current
    (test (>= ?amp .01))                  ;Check battery discharging
    (power cellcur ?val ?amp2 ?min)        ;Get cell current
    (test (<= ?amp2 .01))                  ;Check that cells not lit
  =>
    (assert (in_eclipse ?min))             ;Satellite in eclipse
  )
  (defrule exceed_power_budget
    (power batcur ?val ?amp ?min)         ;Get battery current
    (test (>= ?amp .01))                  ;Check battery discharging
    (power cellcur ?val ?amp2 ?min)        ;Get cell current
    (test (> ?amp2 .01))                  ;Check that cells are lit
  =>
    (assert (pt power_budget 1 exceeded ?min))
    (assert (dec check_power_system))
    (assert (dec do_not_transmit_during_eclipse))
  )
  (defrule delete_old_power_facts
    (done-reading)                        ;No more data
    ?old<-(power ?var ?val ?amp ?min)      ;Get old power fact
  =>
    (retract ?old)                        ;Delete old power fact
  )
  (defrule save_battery_temp
    (tel battemp ?val ?temp ?min)         ;Get battery temp
  =>
    (assert (power battemp ?val ?temp ?min));Save in different form
  )
  (defrule save_battery_volt
    (tel batvolt ?val ?volt ?min)         ;Get battery volt
  =>
    (assert (power batvolt ?val ?volt ?min));Save in different form
  )
  (defrule batt_charged
    (power battemp ?val ?temp ?min)       ;Get battery temp
    (power batvolt ?val ?volt ?min)       ;Get battery voltage
    (test (> (- ?volt (* -.01 ?temp)) 9.8)) ;Test to fit line
  =>
    (assert (event battery ?val charged ?min))
  )
  (defrule bat_temp_increase
    (firsttime bat ?val charged ?min ?after);Get battery charge time
    (power battemp ?val ?temp ?min)       ;Get temperature
    (power battemp ?val ?temp2 ?min2)     ;Get later temperature
    (test (< ?min ?min2))
    (test (< (- ?min2 ?min) 3600))        ;Check in same orbit
    (test (< ?temp ?temp2))              ;Check temp increase
  =>
    (bind ?change (- ?temp2 ?temp))       ;Save temp increase
  )

```

```

(assert (temp_change_charge ?change ?min2))
)
(defrule largest_change
  (temp_change_charge ?change ?min) ;Get a temperature change
  ?less<-(temp_change_charge ?change2 ?min2);Get second change
  (test (< (- ?min2 ?min) 3600)) ;Same orbit
  (test (> ?change ?change2)) ;Which is less
=>
  (retract ?less) ;Del less change
)
(defrule latest_eclipse "Find the last point in eclipse for period"
  (in_eclipse ?min) ;Get old time in eclipse
  ?sec <- (in_eclipse ?min1) ;Get new time in eclipse
  (test (> ?min ?min1)) ;Make sure at later time
=>
  (retract ?sec) ;Delete earlier eclipse time
)
(defrule latest_sun "Find latest time in sun for current orbit"
  (in_sun ?min) ;Get old sun time
  ?sec <- (in_sun ?min1) ;Get second sun time
  (test (> ?min ?min1)) ;Check if sun time later
=>
  (retract ?sec) ;Delete earliest sun time
)
(defrule leave_eclipse "Check satellite has left earth shadow"
  (in_sun ?min) ;Get in sun fact
  ?leave <- (in_eclipse ?min2) ;Get in eclipse fact
  (test (> ?min ?min2)) ;Check in sun after eclipse
=>
  (retract ?leave) ;Delete eclipse fact
  (assert (sun ?min)) ;Store time left shadow
)
(defrule find_latest "Find same data points and delete earliest"
  (latest ?var ?val ?data ?tim)
  ?early<-(latest ?var ?val ?data2 ?tim2)
  (test (> ?tim ?tim2)) ;Check for latest time
=>
  (retract ?early)
)
(defrule solar_cells_cold "Check if solar cells within limits"
  ?new<-(new fact) ;Make sure rule hasn't fired
  (tel cell ?num ?temp ?min) ;Get solar cell temp point
  (test (< ?temp -30)) ;Check if below lower temp
  ?bad<-(counter cell ?num ?count)
=>
  (assert(pt cell ?num cold ?min)) ;Below limits, save problem
  (assert (color_system temp red)) ;Make color of telemetry red
  (assert (color cell ?num red))
  (bind ?counter (+ ?count 1)) ;Increment counter
  (assert (counter cell ?num ?counter))
  (retract ?bad)
  (retract ?new)

```

```

)
(defrule solar_cells_cool "Check solar cells within limits"
  ?new<-(new fact) ;Make sure rule hasn't fired
  (tel cell ?num ?temp ?min) ;Get solar cell temp point
  (test (< ?temp 0)) ;Check if above lower temp
  (test (>= ?temp -30)) ;Check if below normal temp
  ?bad<-(counter cell ?num ?count)
=>
  (assert(pt cell ?num cool ?min)) ;Below limits, save problem
  (assert (color_system temp yellow)) ;Color of telemetry red
  (assert (color cell ?num yellow))
  (bind ?count (+ ?count 1)) ;Increment counter
  (retract ?bad)
  (retract ?new)
  (assert (counter cell ?num ?count))
)
(defrule solar_cells_warm "Check if solar cells within limits"
  ?new<-(new fact) ;Make sure rule hasn't fired
  (tel cell ?num ?temp ?min) ;Get solar cell temp point
  (test (> ?temp 50)) ;Check above normal temp
  (test (<= ?temp 140)) ;Check below upper max temp
  ?bad<-(counter cell ?num ?count)
=>
  (assert(pt cell ?num warm ?min)) ;Above limits, save problem
  (assert (color_system temp yellow)) ;Color telemetry yellow
  (assert (color cell ?num yellow))
  (bind ?count (+ ?count 1)) ;Increment counter
  (retract ?bad)
  (assert (counter cell ?num ?count))
  (retract ?new)
)
(defrule solar_cells_hot "Check if solar cells within limits"
  ?new<-(new fact) ;Make sure rule hasn't fired
  (tel cell ?num ?temp ?min) ;Get solar cell temp point
  (test (> ?temp 140)) ;Check above max temp
  ?bad<-(counter cell ?num ?count)
=>
  (assert(pt cell ?num hot ?min)) ;Save problem
  (assert (color_system temp red)) ;Make color of telemetry red
  (assert (color cell ?num red))
  (bind ?count (+ ?count 1)) ;Increment counter
  (retract ?new)
  (retract ?bad)
  (assert (counter cell ?num ?count))
)
(defrule bat_volt_too_low "Check battery voltage within limits"
  ?new<-(new fact) ;Make sure rule hasn't fired
  (tel batvolt ?num ?volt ?min) ;Get battery voltage point
  (test (< ?volt 10)) ;Check below min volt range
  ?bad<-(counter batvolt ?num ?count)
=>
  (assert(pt batvolt ?num too-low ?min)) ;Save problem

```



```

(assert (color_system power red)) ;Make color of telemetry red
(assert (color batvolt ?num red))
(bind ?count (+ ?count 1)) ;Increment counter
(retract ?bad)
(assert (counter batvolt ?num ?count))
(retract ?new)
(assert (dec check_power_system))
(assert (dec recondition_batteries))
(assert (dec change_operations))
)
(defrule bat_volt_low "Check battery voltage within limits"
?new<-(new fact) ;Make sure rule hasn't fired
(tel batvolt ?num ?volt ?min) ;Get battery voltage point
(test (< ?volt 11.5)) ;Check below min normal volt
(test (>= ?volt 10)) ;Check if above min volt
?bad<-(counter batvolt ?num ?count)
=>
(assert(pt batvolt ?num low ?min)) ;Below limits, save problem
(assert (color_system power yellow)) ;Color of telemetry red
(assert (color batvolt ?num yellow))
(bind ?count (+ ?count 1)) ;Increment counter
(retract ?bad)
(assert (counter batvolt ?num ?count))
(retract ?new)
)
(defrule bat_volt_high "Check battery voltage within limits"
?new<-(new fact) ;Make sure rule hasn't fired
(tel batvolt ?num ?volt ?min) ;Get battery voltage point
(test (> ?volt 13.5)) ;Check above max normal volt
(test (<= ?volt 15)) ;Check if below max volt
?bad<-(counter batvolt ?num ?count)
=>
(assert(pt batvolt ?num high ?min)) ;Save problem
(assert (color_system power yellow)) ;Color of telemetry red
(assert (color batvolt ?num yellow))
(bind ?count (+ ?count 1)) ;Increment counter
(retract ?bad)
(retract ?new)
(assert (counter batvolt ?num ?count))
(assert (dec trickle_charge_batteries))
)
(defrule bat_volt_too_high "Check battery voltage within limits"
?new<-(new fact) ;Make sure rule hasn't fired
(tel batvolt ?num ?volt ?min) ;Get battery voltage point
(test (> ?volt 15)) ;Check above max volt range
?bad<-(counter batvolt ?num ?count)
=>
(assert(pt batvolt ?num too-high ?min)) ;Save problem
(assert (color_system power red)) ;Make color of telemetry red
(assert (color batvolt ?num red))
(bind ?count (+ ?count 1)) ;Increment counter
(retract ?bad)

```

```

(retract ?new)
(assert (counter batvolt ?num ?count))
)
(defrule bat_temp_cold "Check battery temp within limits"
  ?new<-(new fact) ;Make sure rule hasn't fired
  (tel battemp ?num ?temp ?min) ;Get battery temp point
  (test (< ?temp -15)) ;Check below min temp range
  ?bad<-(counter battemp ?num ?count)
=>
  (assert(pt battemp ?num cold ?min));Below limits, save problem
  (assert (color_system temp red)) ;Make color of telemetry red
  (assert (color battemp ?num red))
  (bind ?count (+ ?count 1)) ;Increment counter
  (retract ?bad)
  (retract ?new)
  (assert (counter batvolt ?num ?count))
)
(defrule bat_temp_cool "Check battery temp within limits"
  ?new<-(new fact) ;Make sure rule hasn't fired
  (tel battemp ?num ?temp ?min) ;Get battery temp point
  (test (< ?temp -6.7)) ;Check if below normal temp
  (test (>= ?temp -15)) ;Check above min temp range
  ?bad<-(counter battemp ?num ?count)
=>
  (assert(pt battemp ?num cool ?min));Below limits, save problem
  (assert (color_system temp yellow)) ;Color of telemetry yellow
  (assert (color battemp ?num yellow))
  (bind ?count (+ ?count 1)) ;Increment counter
  (retract ?bad)
  (retract ?new)
  (assert (counter battemp ?num ?count))
)
(defrule bat_temp_warm "Check battery temp within limits"
  ?new<-(new fact) ;Make sure rule hasn't fired
  (tel battemp ?num ?temp ?min) ;Get battery temp point
  (test (> ?temp 26.7)) ;Check if above normal temp
  (test (<= ?temp 50)) ;Check below max temp range
  ?bad<-(counter battemp ?num ?count)
=>
  (assert(pt battemp ?num warm ?min));Above limits, save problem
  (assert (color_system temp yellow)) ;Color of telemetry yellow
  (assert (color battemp ?num yellow))
  (bind ?count (+ ?count 1)) ;Increment counter
  (retract ?bad)
  (retract ?new)
  (assert (counter battemp ?num ?count))
)
(defrule bat_temp_hot "Check battery temp within limits"
  ?new<-(new fact) ;Make sure rule hasn't fired
  (tel battemp ?num ?temp ?min) ;Get battery temp point
  (test (> ?temp 50)) ;Check above max temp range
  ?bad<-(counter battemp ?num ?count)

```

```

=>
  (assert(pt battemp ?num hot ?min)) ;Above limits, save problem
  (assert (color_system temp red)) ;Make color of telemetry red
  (assert (color battemp ?num red))
  (bind ?count (+ ?count 1)) ;Increment counter
  (retract ?bad)
  (assert (counter battemp ?num ?count))
  (retract ?new)
  (assert (dec trickle_charge_batteries))
)
(defrule battery_current_high
  ?new<-(new fact) ;Make sure rule hasn't fired
  (tel batcur ?num ?amp ?min) ;Get battery current
  (test (> ?amp 2))
  (test (<= ?amp 5))
  ?bad<-(counter batcur ?num ?count)
=>
  (assert (pt batcur ?num high ?min))
  (assert (color_system power yellow)) ;Color of telemetry yellow
  (assert (color batcur ?num yellow))
  (bind ?count (+ ?count 1)) ;Increment counter
  (retract ?bad)
  (assert (counter batcur ?num ?count))
  (retract ?new)
)
(defrule battery_current_too_high
  ?new<-(new fact) ;Make sure rule hasn't fired
  (tel batcur ?num ?amp ?min) ;Get battery current
  (test (> ?amp 5))
  ?bad<-(counter batcur ?num ?count)
=>
  (assert (pt batcur ?num too_high ?min))
  (assert (color_system power red)) ;Make color of telemetry red
  (assert (color batcur ?num red))
  (bind ?count (+ ?count 1)) ;Increment counter
  (retract ?bad)
  (assert (counter batcur ?num ?count))
  (retract ?new)
)
(defrule tx_current_high
  ?new<-(new fact) ;Make sure rule hasn't fired
  (tel txcur ?num ?amp ?min) ;Get transmitter current
  (test (> ?amp 2))
  (test (<= ?amp 5))
  ?bad<-(counter txcur ?num ?count)
=>
  (assert (pt txcur ?num high ?min))
  (assert (color_system power yellow)) ;Make color of telemetry red
  (assert (color txcur ?num yellow))
  (bind ?count (+ ?count 1)) ;Increment counter
  (retract ?bad)
  (assert (counter txcur ?num ?count))

```

```

(retract ?new)
)
(defrule tx_current_too_high
  ?new<-(new fact)                ;Make sure rule hasn't fired
  (tel txcur ?num ?amp ?min)       ;Get transmitter current
  (test (> ?amp 5))
  ?bad<-(counter txcur ?num ?count)
=>
  (assert (pt txcur ?num too_high ?min))
  (assert (color_system power red)) ;Make color of telemetry red
  (assert (color txcur ?num red))
  (bind ?count (+ ?count 1))       ;Increment counter
  (retract ?bad)
  (assert (counter txcur ?num ?count))
  (retract ?new)
)
(defrule bus_volt_too_low "Check bus voltage within limits"
  ?new<-(new fact)                ;Make sure rule hasn't fired
  (tel busvolt ?num ?volt ?min)    ;Get bus voltage point
  (test (< ?volt 10))              ;Check if below min volt range
  ?bad<-(counter busvolt ?num ?count)
=>
  (assert(pt busvolt ?num too-low ?min)) ;Save problem
  (assert (color_system power red))      ;Make color of telemetry red
  (assert (color busvolt ?num red))
  (bind ?count (+ ?count 1))             ;Increment counter
  (retract ?bad)
  (assert (counter busvolt ?num ?count))
  (retract ?new)
)
(defrule bus_volt_low "Check bus voltage within limits"
  ?new<-(new fact)                ;Make sure rule hasn't fired
  (tel busvolt ?num ?volt ?min)    ;Get bus voltage point
  (test (< ?volt 11.5))            ;Check below min normal volt
  (test (>= ?volt 10))             ;Check above min volt
  ?bad<-(counter busvolt ?num ?count)
=>
  (assert(pt busvolt ?num low ?min)) ;Below limits, save problem
  (assert (color_system power yellow)) ;Telemetry color yellow
  (assert (color busvolt ?num yellow))
  (bind ?count (+ ?count 1))       ;Increment counter
  (retract ?bad)
  (assert (counter busvolt ?num ?count))
  (retract ?new)
)
(defrule bus_volt_high "Check bus voltage within limits"
  ?new<-(new fact)                ;Make sure rule hasn't fired
  (tel busvolt ?num ?volt ?min)    ;Get bus voltage point
  (test (> ?volt 13.5))            ;Check above max normal volt
  (test (<= ?volt 15))             ;Check if below max volt
  ?bad<-(counter busvolt ?num ?count)
=>

```

```

(assert(pt busvolt ?num high ?min)) ;Save problem
(assert (color_system power yellow)) ;Color of telemetry yellow
(assert (color busvolt ?num yellow))
(bind ?count (+ ?count 1)) ;Increment counter
(retract ?bad)
(retract ?new)
(assert (counter busvolt ?num ?count))
(assert (dec trickle_charge_busteries))
)
(defrule bus_volt_too_high "Check bust voltage within limits"
?new<-(new fact) ;Make sure rule hasn't fired
(tel busvolt ?num ?volt ?min) ;Get bust voltage point
(test (> ?volt 15)) ;Check above max volt range
?bad<-(counter busvolt ?num ?count)
=>
(assert(pt busvolt ?num too-high ?min)) ;Save problem
(assert (color_system power red)) ;Make color of telemetry red
(assert (color busvolt ?num red))
(bind ?count (+ ?count 1)) ;Increment counter
(retract ?bad)
(retract ?new)
(assert (counter busvolt ?num ?count))
)
(defrule bus_temp_cold "Check bus temp within limits"
?new<-(new fact) ;Make sure rule hasn't fired
(tel bustemp ?num ?temp ?min) ;Get bus temp point
(test (< ?temp -10)) ;Check below min temp range
?bad<-(counter bustemp ?num ?count)
=>
(assert(pt bustemp ?num cold ?min));Save problem
(assert (color_system temp red)) ;Make color of telemetry red
(assert (color bustemp ?num red))
(bind ?count (+ ?count 1)) ;Increment counter
(retract ?bad)
(retract ?new)
(assert (counter bustemp ?num ?count))
)
(defrule bus_temp_cool "Check bus temp within limits"
?new<-(new fact) ;Make sure rule hasn't fired
(tel bustemp ?num ?temp ?min) ;Get bus temp point
(test (< ?temp 0)) ;Check if below normal temp
(test (>= ?temp -10)) ;Check above min temp range
?bad<-(counter bustemp ?num ?count)
=>
(assert(pt bustemp ?num cool ?min));Below limits, save problem
(assert (color_system temp yellow)) ;Color of telemetry yellow
(assert (color bustemp ?num yellow))
(bind ?count (+ ?count 1)) ;Increment counter
(retract ?bad)
(assert (counter bustemp ?num ?count))
)
(defrule bus_temp_warm "Check bus temp within limits"

```

```

?new<-(new fact) ;Make sure rule hasn't fired
(tel bustemp ?num ?temp ?min) ;Get bus temp point
(test (> ?temp 40)) ;Check if above normal temp
(test (<= ?temp 50)) ;Check if below max temp range
?bad<-(counter bustemp ?num ?count)
=>
(assert(pt bustemp ?num warm ?min)) ;Save problem
(assert (color_system temp yellow)) ;Make color of telemetry red
(assert (color bustemp ?num yellow))
(bind ?count (+ ?count 1)) ;Increment counter
(retract ?new)
(retract ?bad)
(assert (counter bustemp ?num ?count))
)
(defrule bus_temp_hot "Check bus temp within limits"
?new<-(new fact) ;Make sure rule hasn't fired
(tel bustemp ?num ?temp ?min) ;Get bus temp point
(test (> ?temp 50)) ;Check above max temp range
?bad<-(counter bustemp ?num ?count)
=>
(assert(pt bustemp ?num hot ?min)) ;Save problem
(assert (color_system temp red)) ;Make color of telemetry red
(assert (color bustemp ?num red))
(bind ?count (+ ?count 1)) ;Increment counter
(retract ?bad)
(retract ?new)
(assert (counter bustemp ?num ?count))
)
(defrule tx_volt_too_low "Check tx voltage within limits"
?new<-(new fact) ;Make sure rule hasn't fired
(tel txvolt ?num ?volt ?min) ;Get tx voltage point
(test (< ?volt 10)) ;Check below min volt range
?bad<-(counter txvolt ?num ?count)
=>
(assert(pt txvolt ?num too-low ?min)) ;Save problem
(assert (color_system power red)) ;Make color of telemetry red
(assert (color txvolt ?num red))
(bind ?count (+ ?count 1)) ;Increment counter
(retract ?bad)
(assert (counter txvolt ?num ?count))
(retract ?new)
(assert (dec check_power_system))
)
(defrule tx_volt_low "Check tx voltage within limits"
?new<-(new fact) ;Make sure rule hasn't fired
(tel txvolt ?num ?volt ?min) ;Get tx voltage point
(test (< ?volt 11.5)) ;Check below min normal volt
(test (>= ?volt 10)) ;Check if above min volt
?bad<-(counter txvolt ?num ?count)
=>
(assert(pt txvolt ?num low ?min)) ;Below limits, save problem
(assert (color_system power yellow)) ;Color of telemetry yellow

```

```

(assert (color txvolt ?num yellow))
(bind ?count (+ ?count 1))          ;Increment counter
(retract ?bad)
(assert (counter txvolt ?num ?count))
(retract ?new)
)
(defrule tx_volt_high "Check tx voltage within limits"
  ?new<-(new fact)                  ;Make sure rule hasn't fired
  (tel txvolt ?num ?volt ?min)      ;Get tx voltage point
  (test (> ?volt 13.5))              ;Check above max normal volt
  (test (<= ?volt 15))               ;Check if below max volt
  ?bad<-(counter txvolt ?num ?count)
=>
  (assert(pt txvolt ?num high ?min)) ;Above limits, save problem
  (assert (color_system power yellow)) ;Telemetry color yellow
  (assert (color txvolt ?num yellow))
  (bind ?count (+ ?count 1))          ;Increment counter
  (retract ?bad)
  (retract ?new)
  (assert (counter txvolt ?num ?count))
)
(defrule tx_volt_too_high "Check tx voltage within limits"
  ?new<-(new fact)                  ;Make sure rule hasn't fired
  (tel txvolt ?num ?volt ?min)      ;Get tx voltage point
  (test (> ?volt 15))                ;Check if above max volt range
  ?bad<-(counter txvolt ?num ?count)
=>
  (assert(pt txvolt ?num too-high ?min)) ;Save problem
  (assert (color_system power red))    ;Make color of telemetry red
  (assert (color txvolt ?num red))
  (bind ?count (+ ?count 1))          ;Increment counter
  (retract ?bad)
  (retract ?new)
  (assert (counter txvolt ?num ?count))
)
(defrule tx_temp_cold "Check transmitter temp within limits"
  ?new<-(new fact)                  ;Make sure rule hasn't fired
  (tel txtmp ?num ?temp ?min)        ;Get transmitter temp point
  (test (< ?temp -10))               ;Check if below min temp range
  ?bad<-(counter txtmp ?num ?count)
=>
  (assert(pt txtmp ?num cold ?min)) ;Save problem
  (assert (color_system temp red))    ;Make color of telemetry red
  (assert (color txtmp ?num red))
  (bind ?count (+ ?count 1))          ;Increment counter
  (retract ?new)
  (retract ?bad)
  (assert (counter txtmp ?num ?count))
  (assert (dec transmit_beacon))
)
(defrule tx_temp_cool "Check transmitter temp within limits"
  ?new<-(new fact)                  ;Make sure rule hasn't fired

```

```

    (tel txtemp ?num ?temp ?min)      ;Get transmitter temp point
    (test (< ?temp 0))                 ;Check if below normal temp
    (test (>= ?temp -10))               ;Check above min temp range
    ?bad<-(counter txtemp ?num ?count)

=>
    (assert(pt txtemp ?num cool ?min)) ;Save problem
    (assert (color_system temp yellow)) ;Color of telemetry red
    (assert (color txtemp ?num yellow))
    (bind ?count (+ ?count 1))         ;Increment counter
    (retract ?bad)
    (retract ?new)
    (assert (counter txtemp ?num ?count))
  )
  (defrule tx_temp_warm "Check transmitter temp within limits"
    ?new<-(new fact)                   ;Make sure rule hasn't fired
    (tel txtemp ?num ?temp ?min)       ;Get transmitter temp point
    (test (> ?temp 40))                 ;Check if above normal temp
    (test (<= ?temp 50))                 ;Check if below max temp range
    ?bad<-(counter txtemp ?num ?count)

=>
    (assert(pt txtemp ?num warm ?min)) ;Save problem
    (assert (color_system temp yellow)) ;Make color of telemetry red
    (assert (color txtemp ?num yellow))
    (bind ?count (+ ?count 1))         ;Increment counter
    (retract ?bad)
    (retract ?new)
    (assert (counter txtemp ?num ?count))
  )
  (defrule tx_temp_hot "Check transmitter temp within limits"
    ?new<-(new fact)                   ;Make sure rule hasn't fired
    (tel txtemp ?num ?temp ?min)       ;Get transmitter temp point
    (test(> ?temp 50))                 ;Check above max temp range
    ?bad<-(counter txtemp ?num ?count)

=>
    (assert(pt txtemp ?num hot ?min))  ;Save problem
    (assert (color_system temp red))    ;Make color of telemetry red
    (assert (color txtemp ?num red))
    (bind ?count (+ ?count 1))         ;Increment counter
    (retract ?bad)
    (retract ?new)
    (assert (counter txtemp ?num ?count))
    (assert (dec switch_txmitter))
  )
  (defrule rx_temp_cold "Check receiver temp within limits"
    ?new<-(new fact)                   ;Make sure rule hasn't fired
    (tel rxtemp ?num ?temp ?min)       ;Get receiver temp point
    (test (< ?temp -10))                 ;Check below min temp range
    ?bad<-(counter rxtemp ?num ?count)

=>
    (assert(pt rxtemp ?num cold ?min)) ;Save problem
    (assert (color_system temp red))    ;Make color of telemetry red
    (assert (color rxtemp ?num red))
  )

```



```

(bind ?count (+ ?count 1))      ;Increment counter
(retract ?bad)
(retract ?new)
(assert (counter rxtemp ?num ?count))
)
(defrule rx_temp_cool "Check receiver temp within limits"
  ?new<-(new fact)              ;Make sure rule hasn't fired
  (tel rxtemp ?num ?temp ?min)  ;Get receiver temp point
  (test (< ?temp 0))             ;Check if below normal temp
  (test (>= ?temp -10))          ;Check above min temp range
  ?bad<-(counter rxtemp ?num ?count)
=>
  (assert(pt rxtemp ?num cool ?min)) ;Save problem
  (assert (color_system temp yellow)) ;Color of telemetry yellow
  (assert (color rxtemp ?num yellow))
  (bind ?count (+ ?count 1))      ;Increment counter
  (retract ?bad)
  (retract ?new)
  (assert (counter rxtemp ?num ?count))
)
(defrule rx_temp_warm "Check receiver temp within limits"
  ?new<-(new fact)              ;Make sure rule hasn't fired
  (tel rxtemp ?num ?temp ?min)  ;Get receiver temp point
  (test (> ?temp 40))            ;Check if above normal temp
  (test (<= ?temp 50))            ;Check below max temp range
  ?bad<-(counter rxtemp ?num ?count)
=>
  (assert(pt rxtemp ?num warm ?min)) ;Save problem
  (assert (color_system temp yellow)) ;Color telemetry yellow
  (assert (color rxtemp ?num yellow))
  (bind ?count (+ ?count 1))      ;Increment counter
  (retract ?new)
  (retract ?bad)
  (assert (counter rxtemp ?num ?count))
)
(defrule rx_temp_hot "Check receiver temp within limits"
  ?new<-(new fact)              ;Make sure rule hasn't fired
  (tel rxtemp ?num ?temp ?min)  ;Get receiver temp point
  (test(> ?temp 50))             ;Check above max temp range
  ?bad<-(counter rxtemp ?num ?count)
=>
  (assert(pt rxtemp ?num hot ?min)) ;Save problem
  (assert (color_system temp red))  ;Color of telemetry red
  (assert (color rxtemp ?num red))
  (bind ?count (+ ?count 1))      ;Increment counter
  (retract ?bad)
  (retract ?new)
  (assert (counter rxtemp ?num ?count))
  (assert (dec switch_rcvr))
)
(defrule dcs_temp_cold "Check dcs temp within limits"
  ?new<-(new fact)              ;Make sure rule hasn't fired

```

```

    (tel dcstemp ?num ?temp ?min)      ;Get dcs temp point
    (test (< ?temp -10))                ;Check below min temp range
    ?bad<-(counter dcstemp ?num ?count)
=>
    (assert(pt dcs ?num cold ?min))      ;Save problem
    (assert (color_system temp red))      ;Color of telemetry red
    (assert (color dcstemp ?num red))
    (bind ?count (+ ?count 1))           ;Increment counter
    (retract ?new)
    (retract ?bad)
    (assert (counter dcstemp ?num ?count))
)
(defrule dcs_temp_cool "Check dcs temp within limits"
  ?new<-(new fact)                      ;Make sure rule hasn't fired
  (tel dcstemp ?num ?temp ?min)          ;Get dcs temp point
  (test (< ?temp 0))                    ;Check if below normal temp
  (test (>= ?temp -10))                 ;Check above min temp range
  ?bad<-(counter dcstemp ?num ?count)
=>
  (assert(pt dcstemp ?num cool ?min)) ;Save problem
  (assert (color_system temp yellow)) ;Color of telemetry yellow
  (assert (color dcstemp ?num yellow))
  (bind ?count (+ ?count 1))           ;Increment counter
  (retract ?bad)
  (retract ?new)
  (assert (counter dcstemp ?num ?count))
)
(defrule dcs_temp_warm "Check dcs temp within limits"
  ?new<-(new fact)                      ;Make sure rule hasn't fired
  (tel dcstemp ?num ?temp ?min)          ;Get dcs temp point
  (test(> ?temp 40))                    ;Check if above normal temp
  (test(<= ?temp 50))                   ;Check below max temp range
  ?bad<-(counter dcstemp ?num ?count)
=>
  (assert(pt dcstemp ?num warm ?min)) ;Save problem
  (bind ?count (+ ?count 1))           ;Increment counter
  (retract ?new)
  (retract ?bad)
  (assert (color_system temp yellow)) ;Color of telemetry yellow
  (assert (color dcstemp ?num yellow))
  (assert (counter dcstemp ?num ?count))
)
(defrule dcs_temp_hot "Check dcs temp within limits"
  ?new<-(new fact)                      ;Make sure rule hasn't fired
  (tel dcstemp ?num ?temp ?min)          ;Get dcs temp point
  (test(> ?temp 50))                   ;Check above max temp range
  ?bad<-(counter dcstemp ?num ?count)
=>
  (assert(pt dcstemp ?num hot ?min)) ;Save problem
  (assert (color_system temp red))      ;Color of telemetry red
  (assert (color dcstemp ?num red))
  (bind ?count (+ ?count 1))           ;Increment counter

```

```

(retract ?bad)
(retract ?new)
(assert (counter dcstamp ?num ?count))
(assert (dec switch_dcs))
)
(defrule edacseu_counter "Check on the frequency of seu in cpu."
  (tel edacseucount ?num ?count ?min) ;Find edac counter
  (test (> ?count 1)) ;Test if greater than limit
=>
  (assert (dec increase_ramwash_frequency)) ;Save decision
)
(defrule worst_case_system "Save worst colors for system"
  (color_system ?system red) ;Find case where system is red
  ?better <- (color_system ?system yellow);Find system is yellow
=>
  (retract ?better) ;Delete yellow case
)
(defrule worst_case_tel_pt "Find colors for telemetry and save"
  (color ?telpt ?num red) ;Find where telemetry red
  ?better <- (color ?telpt ?num yellow) ;Find telemetry yellow
=>
  (retract ?better) ;Delete yellow case
)
(defrule initialize_occurence_problem "Start algorithm to find first
occurence of problem per orbit"
  (pt ?telpt ?num ?prob ?min) ;Get telemetry problem
  (sun ?tim) ;Get time leaving eclipse
  (test (>= ?min ?tim)) ;Make time before problem
=>
  (bind ?after (- ?min ?tim)) ;Determine when
  (assert (firsttime ?telpt ?num ?prob ?min ?after)) ;Make first
)
(defrule find_first_occurence "Find the first time that a problem
occurred"
  (firsttime ?telpt ?num ?prob ?min ?after) ;Get firsttime fact
  ?sec <- (firsttime ?telpt ?num ?prob ?min1 ?after1);Get second
  (sun ?time) ;Find time left eclipse
  (test (< ?min ?min1)) ;Check first is earlier
  (test (> ?min ?time)) ;Check first after eclipse
  (test (> ?min1 ?time)) ;Check second after eclipse
=>
  (retract ?sec) ;Delete later fact
)
(defrule initialize_high_and_low_orbit "Find high and low values"
  (tel ?var ?val ?data ?tim) ;Get telemetry point
  (sun ?tim1) ;Find if left eclipse
  (test (>= ?tim ?tim1)) ;Check tel pt after eclipse
  (tel ~not ?num ?prob ?min) ;Do not find high value for
  (tel ~edacseutime ?num ?prob ?min) ;counters
  (tel ~unauthatt ?num ?prob ?min)
  (tel ~authlogins ?num ?prob ?min)
  (tel ~attenset ?num ?prob ?min)

```

```

(tel ~edacseucountstart ?num ?prob ?min)
(tel ~RAMwash ?num ?prob ?min)
(tel ~poolbuffers ?num ?prob ?min)
(tel ~edacseucount ?num ?prob ?min)
(tel ~verset ?num ?prob ?min)
(tel ~dcsset ?num ?prob ?min)
(tel ~modeset ?num ?prob ?min)
(tel ~txset ?num ?prob ?min)
(tel ~rxset ?num ?prob ?min)
(tel ~storageused ?num ?prob ?min)
(tel ~sentmail ?num ?prob ?min)
(tel ~rxcdmail ?num ?prob ?min)
(tel ~storedmail ?num ?prob ?min)
(tel ~logins ?num ?prob ?min)
(tel ~logouts ?num ?prob ?min)
(tel ~UInum ?num ?prob ?min)
(tel ~FRMRnum ?num ?prob ?min)
(tel ~UANum ?num ?prob ?min)
(tel ~SABMnum ?num ?prob ?min)
(tel ~REJnum ?num ?prob ?min)
(tel ~DMnum ?num ?prob ?min)
(tel ~RMRnum ?num ?prob ?min)
(tel ~DISCnum ?num ?prob ?min)
(tel ~RRnum ?num ?prob ?min)
(tel ~Inum ?num ?prob ?min)
(tel ~dataout ?num ?prob ?min)
(tel ~datain ?num ?prob ?min)
(tel ~uptime ?num ?prob ?min)
=>
  (assert (high ?var ?val ?data ?tim)) ;Initialize high per orbit
  (assert (highest ?var ?val ?data ?tim)) ;Initialize highest
  (assert (low ?var ?val ?data ?tim)) ;Initialize low per orbit
  (assert (lowest ?var ?val ?data ?tim)) ;Initialize lowest
)
(defrule find_high_orbit "Find the highest value per orbit of each
telemetry point"
  (high ?var ?val ?data ?tim) ;Get a high fact
  ?lower <- (high ?var ?val ?data2 ?tim2);Get a second high fact
  (test (<> ?tim ?tim2)) ;Check not the same
  (test (>= ?data ?data2)) ;Check first is higher
=>
  (retract ?lower) ;Delete second fact
)
(defrule find_highest "Find the highest value in the file for each
telemetry point"
  (highest ?var ?val ?data ?tim) ;Get a highest fact
  ?lower <- (highest ?var ?val ?data2 ?tim2) ;Get second highest
  (test (<> ?tim ?tim2)) ;Check facts not the same
  (test (>= ?data ?data2)) ;Check first fact is higher
=>
  (retract ?lower) ;Delete second fact
)

```

```

(defrule find_low_orbit "Find the lowest value per orbit of each
telemetry point"
  ?lower <- (low ?var ?val ?data ?tim) ;Get a low fact
  ?higher <- (low ?var ?val ?data2 ?tim2) ;Get a second low fact
  (test (<> ?tim ?tim2)) ;Check facts not the same
  (test (<= ?data ?data2)) ;Check first is lower
=>
  (retract ?higher) ;Delete second fact
)
(defrule find_lowest "Find the lowest value of each telemetry point"
  (lowest ?var ?val ?data ?tim) ;Get a low fact
  ?higher <- (lowest ?var ?val ?data2 ?tim2) ;Get second low fact
  (test (<> ?tim ?tim2)) ;Check not the same
  (test (<= ?data ?data2)) ;Check first fact is lower
=>
  (retract ?higher) ;Delete second fact
)
(defrule check_configuration "Check to ensure spacecraft configuration
is the same as the ground station expects"
  (config ?set ?val) ;Get configuration fact
  (tel ?set ?num ?act ?time) ;Get telemetry fact
  (test (<> ?val ?act)) ;Check settings are same
=>
  (assert (pt ?set ?num not_configured ?time)) ;config error
)
(defrule del_config
  (done-reading) ;No more data
  ?con<-(config ?set ?val) ;Get config fact
=>
  (retract ?con) ;Delete config fact
)
(defrule del_high
  (done-reading) ;No more data
  ?hi<-(high ?point ?num ?data ?time);get high fact
=>
  (retract ?hi) ;Delete high fact
)
(defrule del_low
  (done-reading) ;No more data
  ?lo<-(low ?point ?num ?data ?time);Get low fact
=>
  (retract ?lo) ;Delete low fact
)
(defrule constant_out_counter
  (pt ?var ?num ?prob ?tim) ;Find problem fact
  (not (done-reading)) ;More data
  (not (count_out ?var ?num ?count)) ;Problem does not have count
=>
  (assert (count_out ?var ?num 0));Start counter
)
(defrule out_count
  ?next<-(next ?var ?num) ;Get next fact

```

```

?oldcnt<-(count_out ?var ?num ?count);If counter, increment
=>
  (bind ?count (+ ?count 1))
  (assert (count_out ?var ?num ?count))
  (retract ?oldcnt)
  (retract ?next)
)
(defrule check_counts
  (done-reading) ;No more data
  ?cnt<-(count_out ?var ?num ?count);Get count since bad
  (counter ?var ?num ?countbad);Get count out of limits
  (test (= ?count ?countbad));See if same
=>
  (retract ?cnt) ;Delete old count fact
  (assert (problem ?var ?num constantly_out_limits))
)
(defrule not_always_out
  (done-reading) ;No more data
  ?cnt<-(count_out ?var ?num ?count);Get count since bad
  (counter ?var ?num ?countbad);Get count out of limits
  (test (<> ?count ?countbad));See if not same
=>
  (retract ?cnt)
)
(defrule del_next
  (done-reading) ;No more data
  ?next<-(next ?var ?num) ;Get next fact
=>
  (retract ?next) ;Delete next fact
)
(defrule save_first_problems "Save problems found in telemetry"
  (done-reading) ;Find if data files open
  ?prob<-(pt ?var ?val ?data ?tim) ;Get problem fact
  ?when<-(firsttime ?var ?val ?data ?tim ?after)
=>
  (printout problems ?var " " ?val " " ?data " " ?tim " occurred first
time at " ?after " seconds after eclipse" crlf) ;Send file
  (retract ?prob)
  (retract ?when)
)
(defrule save_problems "Save problems found in telemetry"
  (done-reading) ;Find if data files open
  ?prob<-(pt ?var ?val ?data ?tim) ;Get problem fact
  (not (firsttime ?var ?val ?data ?tim ?after))
=>
  (printout problems ?var " " ?val " " ?data " " ?tim crlf)
  (retract ?prob)
)
(defrule save_decisions "Save decisions determined by rules"
  (done-reading) ;Find if data files open
  ?decide<-(dec $?dec) ;Get decision fact
=>

```

```

(printout decide ?dec crlf)          ;Send to decision file
(retract ?decide)
)
(defrule save_telemetry_colors "Save colors determined by rules"
  (done-reading)                    ;Find if data files open
  ?col<-(color ?var ?num ?color)    ;Get color fact
=>
  (printout colors ?var " ?num" "?color crlf) ;Send color file
  (retract ?col)
)
(defrule save_system_colors "Save colors determined by rules"
  (done-reading)                    ;Find if data files open
  ?col<-(color_system ?sys ?color)  ;Get color fact
=>
  (printout colorsys ?sys " ?color crlf) ;Send system color file
  (retract ?col)
)
(defrule save_hi_and_low_data "Save high and low per orbit data"
  ?early<-(sun ?time)              ;Get sun time
  (sun ?time2)                      ;Get second
  (test (< ?time ?time2))          ;Determine earliest
  ?cellhi1<-(high cell 1 ?d1 ?tim1)
  ?cellhi2<-(high cell 2 ?d2 ?tim2)
  ?cellhi3<-(high cell 3 ?d3 ?tim3)
  ?cellhi4<-(high cell 4 ?d4 ?tim4)
  ?cellhi5<-(high cell 5 ?d5 ?tim5)
  ?cellhi6<-(high cell 6 ?d6 ?tim6)
  ?cellhi7<-(high cell 7 ?d7 ?tim7)
  ?cellhi8<-(high cell 8 ?d8 ?tim8)
  ?cellhi9<-(high cell 9 ?d9 ?tim9)
  ?cellhi10<-(high cell 10 ?d10 ?tim10)
  ?cellhi11<-(high cell 11 ?d11 ?tim11)
  ?cellhi12<-(high cell 12 ?d12 ?tim12)
  ?cellhi13<-(high cell 13 ?d13 ?tim13)
  ?cellhi14<-(high cell 14 ?d14 ?tim14)
  ?cellhi15<-(high cell 15 ?d15 ?tim15)
  ?cellhi16<-(high cell 16 ?d16 ?tim16)
  ?cellhi17<-(high cell 17 ?d17 ?tim17)
  ?batthi1<-(high battemp 1 ?d18 ?tim18)
  ?batthi2<-(high battemp 2 ?d19 ?tim19)
  ?batvhi1<-(high batvolt 1 ?d20 ?tim20)
  ?batvhi2<-(high batvolt 2 ?d21 ?tim21)
  ?batihi1<-(high batcur 1 ?d22 ?tim22)
  ?batihi2<-(high batcur 2 ?d23 ?tim23)
  ?txthi1<-(high txtemp 1 ?d24 ?tim24)
  ?txthi2<-(high txtemp 2 ?d25 ?tim26)
  ?rxthi1<-(high rxtemp 1 ?d26 ?tim26)
  ?rxthi2<-(high rxtemp 2 ?d27 ?tim27)
  ?busthi1<-(high bustemp 1 ?d28 ?tim28)
  ?busvhi1<-(high busvolt 1 ?d30 ?tim30)
  ?celllo1<-(low cell 1 ?d32 ?tim32)
  ?celllo2<-(low cell 2 ?d33 ?tim33)

```

```

?celllo3<-(low cell 3 ?d34 ?tim34)
?celllo4<-(low cell 4 ?d35 ?tim35)
?celllo5<-(low cell 5 ?d36 ?tim36)
?celllo6<-(low cell 6 ?d37 ?tim37)
?celllo7<-(low cell 7 ?d38 ?tim38)
?celllo8<-(low cell 8 ?d39 ?tim39)
?celllo9<-(low cell 9 ?d40 ?tim40)
?celllo10<-(low cell 10 ?d41 ?tim41)
?celllo11<-(low cell 11 ?d42 ?tim42)
?celllo12<-(low cell 12 ?d43 ?tim43)
?celllo13<-(low cell 13 ?d44 ?tim44)
?celllo14<-(low cell 14 ?d45 ?tim45)
?celllo15<-(low cell 15 ?d46 ?tim46)
?celllo16<-(low cell 16 ?d47 ?tim47)
?celllo17<-(low cell 17 ?d48 ?tim48)
?battlo1<-(low battemp 1 ?d49 ?tim49)
?battlo2<-(low battemp 2 ?d50 ?tim50)
?batvlo1<-(low batvolt 1 ?d51 ?tim51)
?batvlo2<-(low batvolt 2 ?d52 ?tim52)
?batilo1<-(low batcur 1 ?d53 ?tim53)
?batilo2<-(low batcur 2 ?d54 ?tim54)
?txtlo1<-(low txttemp 1 ?d55 ?tim55)
?txtlo2<-(low txttemp 2 ?d56 ?tim56)
?rxvlo1<-(low rxtemp 1 ?d57 ?tim57)
?rxvlo2<-(low rxtemp 2 ?d58 ?tim58)
?bustlo1<-(low bustemp 1 ?d59 ?tim59)
?busvlo1<-(low busvolt 1 ?d61 ?tim61)
=>
(printout hivals ?time" ?d1" ?d2" ?d3" ?d4" ?d5" ?d6" ?d7"
?d8" ?d9" ?d10" ?d11" ?d12" ?d13" ?d14" ?d15" ")
(printout hivals ?d16" ?d17" ?d18" ?d19" ?d20" ?d21" ?d22"
?d23" ?d24" ?d25" ?d26" ?d27" ?d28" ?d30 crlf)
(retract ?cellhi1) (retract ?cellhi2) (retract ?cellhi3)
(retract ?cellhi4) (retract ?cellhi5) (retract ?cellhi6)
(retract ?cellhi7) (retract ?cellhi8) (retract ?cellhi9)
(retract ?cellhi10) (retract ?cellhi11) (retract ?cellhi12)
(retract ?cellhi13) (retract ?cellhi14) (retract ?cellhi15) (retract
?cellhi16) (retract ?cellhi17) (retract ?batthi1) (retract ?batthi2)
(retract ?batvhi1) (retract ?batvhi2)
(retract ?batihi1) (retract ?batihi2) (retract ?txthi1)
(retract ?txthi2) (retract ?rxthi1) (retract ?rxthi2)
(retract ?busthi1) (retract ?busvhi1)
(printout lovals ?time" ?d32" ?d33" ?d34" ?d35" ?d36" ?d37"
?d38" ?d39" ?d40" ?d41" ?d42" ?d43" ?d44" ?d45" ?d46" ?d47"
?d48" ?d49" ?d50" ?d51" ?d52" ")
(printout lovals ?d53" ?d54" ?d55" ?d56" ?d57" ?d58" ?d59" ?d61
crlf) ;send to data file
(retract ?celllo1) (retract ?celllo2) (retract ?celllo3)
(retract ?celllo4) (retract ?celllo5) (retract ?celllo6)
(retract ?celllo7) (retract ?celllo8) (retract ?celllo9)
(retract ?celllo10) (retract ?celllo11) (retract ?celllo12)

```



```

(retract ?celllo13) (retract ?celllo14) (retract ?celllo15) (retract
?celllo16) (retract ?celllo17) (retract ?battlo1) (retract ?battlo2)
(retract ?batvlo1) (retract ?batvlo2)
(retract ?batilo1) (retract ?batilo2) (retract ?txtlo1)
(retract ?txtlo2) (retract ?rxtilo1) (retract ?rxtilo2)
(retract ?bustlo1) (retract ?busvlo1) (retract ?early)
)
(defrule save_telemetry_data
  (done-reading) ;Find if data files open
  (not (count_out ?var ?num ?count)) ;Constant out done
  ?cellhi1<-(highest cell 1 ?d1 ?tim1)
  ?cellhi2<-(highest cell 2 ?d2 ?tim2)
  ?cellhi3<-(highest cell 3 ?d3 ?tim3)
  ?cellhi4<-(highest cell 4 ?d4 ?tim4)
  ?cellhi5<-(highest cell 5 ?d5 ?tim5)
  ?cellhi6<-(highest cell 6 ?d6 ?tim6)
  ?cellhi7<-(highest cell 7 ?d7 ?tim7)
  ?cellhi8<-(highest cell 8 ?d8 ?tim8)
  ?cellhi9<-(highest cell 9 ?d9 ?tim9)
  ?cellhi10<-(highest cell 10 ?d10 ?tim10)
  ?cellhi11<-(highest cell 11 ?d11 ?tim11)
  ?cellhi12<-(highest cell 12 ?d12 ?tim12)
  ?cellhi13<-(highest cell 13 ?d13 ?tim13)
  ?cellhi14<-(highest cell 14 ?d14 ?tim14)
  ?cellhi15<-(highest cell 15 ?d15 ?tim15)
  ?cellhi16<-(highest cell 16 ?d16 ?tim16)
  ?cellhi17<-(highest cell 17 ?d17 ?tim17)
  ?batthi1<-(highest battemp 1 ?d18 ?tim18)
  ?batthi2<-(highest battemp 2 ?d19 ?tim19)
  ?batvhi1<-(highest batvolt 1 ?d20 ?tim20)
  ?batvhi2<-(highest batvolt 2 ?d21 ?tim21)
  ?batihi1<-(highest batcur 1 ?d22 ?tim22)
  ?batihi2<-(highest batcur 2 ?d23 ?tim23)
  ?txthi1<-(highest txttemp 1 ?d24 ?tim24)
  ?txthi2<-(highest txttemp 2 ?d25 ?tim26)
  ?rxthi1<-(highest rxtemp 1 ?d26 ?tim26)
  ?rxthi2<-(highest rxtemp 2 ?d27 ?tim27)
  ?busthi1<-(highest bustemp 1 ?d28 ?tim28)
  ?busvhi1<-(highest busvolt 1 ?d30 ?tim30)
  ?celllo1<-(lowest cell 1 ?d32 ?tim32)
  ?celllo2<-(lowest cell 2 ?d33 ?tim33)
  ?celllo3<-(lowest cell 3 ?d34 ?tim34)
  ?celllo4<-(lowest cell 4 ?d35 ?tim35)
  ?celllo5<-(lowest cell 5 ?d36 ?tim36)
  ?celllo6<-(lowest cell 6 ?d37 ?tim37)
  ?celllo7<-(lowest cell 7 ?d38 ?tim38)
  ?celllo8<-(lowest cell 8 ?d39 ?tim39)
  ?celllo9<-(lowest cell 9 ?d40 ?tim40)
  ?celllo10<-(lowest cell 10 ?d41 ?tim41)
  ?celllo11<-(lowest cell 11 ?d42 ?tim42)
  ?celllo12<-(lowest cell 12 ?d43 ?tim43)
  ?celllo13<-(lowest cell 13 ?d44 ?tim44)

```

```

?celllo14<-(lowest cell 14 ?d45 ?tim45)
?celllo15<-(lowest cell 15 ?d46 ?tim46)
?celllo16<-(lowest cell 16 ?d47 ?tim47)
?celllo17<-(lowest cell 17 ?d48 ?tim48)
?battlo1<-(lowest battemp 1 ?d49 ?tim49)
?battlo2<-(lowest battemp 2 ?d50 ?tim50)
?batvlo1<-(lowest batvolt 1 ?d51 ?tim51)
?batvlo2<-(lowest batvolt 2 ?d52 ?tim52)
?batilo1<-(lowest batcur 1 ?d53 ?tim53)
?batilo2<-(lowest batcur 2 ?d54 ?tim54)
?txtlo1<-(lowest txttemp 1 ?d55 ?tim55)
?txtlo2<-(lowest txttemp 2 ?d56 ?tim56)
?rxvlo1<-(lowest rxtemp 1 ?d57 ?tim57)
?rxvlo2<-(lowest rxtemp 2 ?d58 ?tim58)
?bustlo1<-(lowest bustemp 1 ?d59 ?tim59)
?busvlo1<-(lowest busvolt 1 ?d61 ?tim61)
?cellcnt1<-(counter cell 1 ?dc32)
?cellcnt2<-(counter cell 2 ?dc33)
?cellcnt3<-(counter cell 3 ?dc34)
?cellcnt4<-(counter cell 4 ?dc35)
?cellcnt5<-(counter cell 5 ?dc36)
?cellcnt6<-(counter cell 6 ?dc37)
?cellcnt7<-(counter cell 7 ?dc38)
?cellcnt8<-(counter cell 8 ?dc39)
?cellcnt9<-(counter cell 9 ?dc40)
?cellcnt10<-(counter cell 10 ?dc41)
?cellcnt11<-(counter cell 11 ?dc42)
?cellcnt12<-(counter cell 12 ?dc43)
?cellcnt13<-(counter cell 13 ?dc44)
?cellcnt14<-(counter cell 14 ?dc45)
?cellcnt15<-(counter cell 15 ?dc46)
?cellcnt16<-(counter cell 16 ?dc47)
?cellcnt17<-(counter cell 17 ?dc48)
?battcnt1<-(counter battemp 1 ?dc49)
?battcnt2<-(counter battemp 2 ?dc50)
?batvcnt1<-(counter batvolt 1 ?dc51)
?batvcnt2<-(counter batvolt 2 ?dc52)
?baticnt1<-(counter batcur 1 ?dc53)
?baticnt2<-(counter batcur 2 ?dc54)
?txtcnt1<-(counter txttemp 1 ?dc55)
?txtcnt2<-(counter txttemp 2 ?dc56)
?rxvcnt1<-(counter rxtemp 1 ?dc57)
?rxvcnt2<-(counter rxtemp 2 ?dc58)
?bustcnt1<-(counter bustemp 1 ?dc59)
?busvcnt1<-(counter busvolt 1 ?dc61)
?celllast1<-(latest cell 1 ?dl32 ?timdl32)
?celllast2<-(latest cell 2 ?dl33 ?timdl33)
?celllast3<-(latest cell 3 ?dl34 ?timdl34)
?celllast4<-(latest cell 4 ?dl35 ?timdl35)
?celllast5<-(latest cell 5 ?dl36 ?timdl36)
?celllast6<-(latest cell 6 ?dl37 ?timdl37)
?celllast7<-(latest cell 7 ?dl38 ?timdl38)

```

```

?celllast8<-(latest cell 8 ?dl39 ?timdl39)
?celllast9<-(latest cell 9 ?dl40 ?timdl40)
?celllast10<-(latest cell 10 ?dl41 ?timdl41)
?celllast11<-(latest cell 11 ?dl42 ?timdl42)
?celllast12<-(latest cell 12 ?dl43 ?timdl43)
?celllast13<-(latest cell 13 ?dl44 ?timdl44)
?celllast14<-(latest cell 14 ?dl45 ?timdl45)
?celllast15<-(latest cell 15 ?dl46 ?timdl46)
?celllast16<-(latest cell 16 ?dl47 ?timdl47)
?celllast17<-(latest cell 17 ?dl48 ?timdl48)
?battlast1<-(latest battemp 1 ?dl49 ?timdl49)
?battlast2<-(latest battemp 2 ?dl50 ?timdl50)
?batvlast1<-(latest batvolt 1 ?dl51 ?timdl51)
?batvlast2<-(latest batvolt 2 ?dl52 ?timdl52)
?batilast1<-(latest batcur 1 ?dl53 ?timdl53)
?batilast2<-(latest batcur 2 ?dl54 ?timdl54)
?txtlast1<-(latest txtemp 1 ?dl55 ?timdl55)
?txtlast2<-(latest txtemp 2 ?dl56 ?timdl56)
?rxlast1<-(latest rxtemp 1 ?dl57 ?timdl57)
?rxlast2<-(latest rxtemp 2 ?dl58 ?timdl58)
?bustlast1<-(latest bustemp 1 ?dl59 ?timdl59)
?busvlast1<-(latest busvolt 1 ?dl61 ?timdl61)
=>
(printout teldata ?d1" "?d32" "?dc32" "?dl32 crlf ?d2" "?d33" "?dc33"
"?dl33 crlf ?d3" "?d34" ")
(printout teldata ?dc34" "?dl34 crlf ?d4" "?d35" "?dc35" "?dl35 crlf
?d5" "?d36" "?dc36" "?dl36)
(printout teldata crlf ?d6" "?d37" "?dc37" "?dl37 crlf ?d7" "?d38"
"?dc38" "?dl38 crlf ?d8" "?d39)
(printout teldata " "?dc39" "?dl39 crlf ?d9" "?d40" "?dc40" "?dl40
crlf ?d10" "?d41" "?dc41" "?dl41)
(printout teldata crlf ?d11" "?d42" "?dc42" "?dl42 crlf ?d12"
"?d43" "?dc43" "?dl43 crlf ?d13" "?d44)
(printout teldata " "?dc44" "?dl44 crlf ?d14" "?d45" "?dc45" "?dl45
crlf ?d15" "?d46" "?dc46" "?dl46 crlf )
(printout teldata ?d16" "?d47" "?dc47" "?dl47 crlf ?d17" "?d48"
"?dc48" "?dl48 crlf ?d18" "?d49" ")
(printout teldata ?dc49" "?dl49 crlf ?d19" "?d50" "?dc50" "?dl50 crlf
?d20" "?d51" "?dc51" "?dl51 crlf)
(printout teldata ?d21" "?d52" "?dc52" "?dl52 crlf ?d22" "?d53"
"?dc53" "?dl53 crlf ?d23" "?d54" ")
(printout teldata ?dc54" "?dl54 crlf ?d24" "?d55" "?dc55" "?dl55 crlf
?d25" "?d56" "?dc56" "?dl56 crlf )
(printout teldata ?d26" "?d57" "?dc57" "?dl57 crlf ?d27" "?d58"
"?dc58" "?dl58 crlf ?d28 " "?d59" ")
(printout teldata ?dc59" "?dl59 crlf ?d30" "?d61" "?dc61" "?dl61
crlf)
(retract ?cellhi1) (retract ?cellhi2) (retract ?cellhi3)
(retract ?cellhi4) (retract ?cellhi5) (retract ?cellhi6)
(retract ?cellhi7) (retract ?cellhi8) (retract ?cellhi9)
(retract ?cellhi10) (retract ?cellhi11) (retract ?cellhi12) (retract
?cellhi13) (retract ?cellhi14) (retract ?cellhi15) (retract

```

?cellhi16) (retract ?cellhi17) (retract ?batthi1) (retract ?batthi2)
 (retract ?batvhi1) (retract ?batvhi2)
 (retract ?batihi1) (retract ?batihi2) (retract ?txthi1)
 (retract ?txthi2) (retract ?rxthi1) (retract ?rxthi2)
 (retract ?busthi1) (retract ?busvhi1) (retract ?celllo1)
 (retract ?celllo2) (retract ?celllo3) (retract ?celllo4)
 (retract ?celllo5) (retract ?celllo6) (retract ?celllo7)
 (retract ?celllo8) (retract ?celllo9) (retract ?celllo10) (retract
 ?celllo11) (retract ?celllo12) (retract ?celllo13) (retract
 ?celllo14) (retract ?celllo15) (retract ?celllo16) (retract
 ?celllo17) (retract ?battlo1) (retract ?battlo2)
 (retract ?batvlo1) (retract ?batvlo2) (retract ?batilo1)
 (retract ?batilo2) (retract ?txtlo1) (retract ?txtlo2)
 (retract ?rxvlo1) (retract ?rxvlo2) (retract ?bustlo1)
 (retract ?busvlo1) (retract ?cellcnt1) (retract ?cellcnt2)
 (retract ?cellcnt3) (retract ?cellcnt4) (retract ?cellcnt5) (retract
 ?cellcnt6) (retract ?cellcnt7) (retract ?cellcnt8) (retract
 ?cellcnt9) (retract ?cellcnt10) (retract ?cellcnt11) (retract
 ?cellcnt12) (retract ?cellcnt13) (retract ?cellcnt14)
 (retract ?cellcnt15) (retract ?cellcnt16) (retract ?cellcnt17)
 (retract ?battcnt1) (retract ?battcnt2) (retract ?batvcnt1) (retract
 ?batvcnt2) (retract ?baticnt1) (retract ?baticnt2) (retract ?txtcnt1)
 (retract ?txtcnt2) (retract ?rxvcnt1)
 (retract ?rxvcnt2) (retract ?bustcnt1) (retract ?busvcnt1) (retract
 ?celllast1) (retract ?celllast2) (retract ?celllast3) (retract
 ?celllast4) (retract ?celllast5) (retract ?celllast6) (retract
 ?celllast7) (retract ?celllast8) (retract ?celllast9)
 (retract ?celllast10) (retract ?celllast11) (retract ?celllast12)
 (retract ?celllast13) (retract ?celllast14) (retract ?celllast15)
 (retract ?celllast16) (retract ?celllast17) (retract ?battlast1)
 (retract ?battlast2) (retract ?batvlast1) (retract ?batvlast2)
 (retract ?batilast1) (retract ?batilast2) (retract ?txtlast1)
 (retract ?txtlast2) (retract ?rxlast1) (retract ?rxlast2) (retract
 ?bustlast1) (retract ?busvlast1)
)

APPENDIX C.

General User Commands

LIST \B\U\A\CS "callsign" transmit list of message and bulletins
 \B list all bulletins
 \U list all messages to user callsign
 \A list all bulletins and messages
 \CS "callsign" list all messages to callsign other than users'
 default- list all bulletins and messages to user callsign

READ \B,#\U,#\A\CS "callsign", # read messages and bulletins
 \B read all bulletins
 \U read all messages to user callsign
 \A read all bulletins and messages
 \CS "callsign" read all messages to callsign other than users'
 ,# read only message or bulletin indicated by number
 default- read all bulletins and messages to user callsign

DEL \# delete messages to user callsign
 \# delete message indicated by number

SEND \"callsign\" \"filename\" send a message to another user
 \"callsign\" callsign of person to receive message
 \"filename\" file that contains message to other callsign

LOGON \"callsign\" initialize communications link with satellite
 \"callsign\" official callsign of user

LOGOFF end session with satellite and break communications link

READ TEL read current telemetry in memory

Ground Control Commands

Includes those for the general user plus:

DUMP LIST \B\M\A\CS "callsign"	transmit list of messages and bulletins \B list all bulletins \M list all messages \A list all bulletins and messages \CS "callsign" list all messages to callsign default- list all bulletins and messages
DUMP MAIL \B\M\CS "callsign"	transmit messages and bulletins \B send all bulletins \M send all messages \A send all bulletins and messages \CS "callsign" send all messages to callsign default- send all bulletins and messages
POST BULLETIN \"filename"	post a bulletin for all users to access \"filename" file that contains bulletin
REMOVE BULLETIN \#T<time>	remove a bulletin from the spacecraft memory \# remove bulletin indicated by number \T<time> remove bulletin at specified time
PURGE MAIL \CS "callsign",#T<time>\A	delete mail \CS "callsign" delete mail to callsign \T<time> delete mail at specified time \A delete all mail ,# delete message indicated by number default- delete all mail to user callsign
READ CUR TEL	read current telemetry in memory
READ STOR TEL	download stored telemetry file for analysis
DEL STOR TEL	delete information stored in telemetry file
READ OS PAR	read the parameters of the operating system, pointers, etc.
READ DATA \#	read data values in memory location \# memory location indicated by number

READ CLOCK	read spacecraft time
SUPER \<password>	enter the super user mode \<password> is a function of the day of satellite operation, will be found in a table format. Allows the satellite to execute subsystem commands.
EXIT	exit the super user mode. Satellite will no longer accept subsystem commands from the ground station callsign.
READ CMDS	read time tagged command buffer
READ EVENTS	read event log
Subsystem Commands	
These commands require that the user has executed SUPER command.	
CLR CMDS	clear time tagged command buffer
CLR EVENTS	clear event log
SET CLOCK \<time>	set the spacecraft clock \<time> time to set as soon as received
LD SOFT \"filename"	send request to load software with size of code. \"filename\" file that contains new software, has size information in file default- load new software and wait for command to begin executing new software
RUN SOFT \"address"	begin operations using new software \"address\" beginning address of new software to run
DEL SOFT \"address\" \"size"	delete software at load address \"address\" location of beginning of software to be deleted \"size\" size of software to be deleted

SET TEL \ "filename" \ T <time>	set polling rates for the satellite telemetry \ "filename" file containing the new polling rates \ T <time> time to implement new polling rates default- implement new polling rates as soon as received
DISCH BATT \ A \ B	Discharge battery A or B. Only one can discharge at a time.
TRKL BATT \ A \ B	trickle charge battery \ A \ B trickle charge battery A or B as selected default- trickle charge both batteries
CHRG BATT \ A \ B	charge battery \ A \ B charge battery A or B as selected default- charge both batteries
PROC \ A \ B	set processor \ A \ B set processor A or B to be operational default- keep same processor operating, send which processor is operating
RF \ A \ B	Set RF selected \ A \ B set rf section A or B to be operational default- keep same rf section operating, send which rf section selected
MAILBOX \ A \ B	Set Mailbox to be used \ A \ B set mailbox A or B to be operational default- keep same mailbox operational, send which mailbox selected.
MUX \ A \ B	Set multiplexer selected \ A \ B set multiplexer A or B to be operational default- keep same multiplexer operational, send which multiplexer selected.
MAX TX	set current transmitter to max power out

MODE \BPSK\SS

set communications mode

\BPSK select BPSK mode for communications

\SS select spread spectrum for communications

default- keep same communications mode, send
which communications mode satellite is in

ATTEN \#

set the attenuation on the current transmitter

\# select the level 1-8 to change the attenuation level

default- keep attenuation levels the same, send
attenuation level

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Dudley Knox Library, Code 052 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chairman, (Code SP) Space Systems Academic Group Naval Postgraduate School Monterey, CA 93943-5000	1
4.	I. Michael Ross, (Code AA) Naval Postgraduate School Monterey, CA 93943-5000	6
5.	Neil C. Rowe, (Code CS) Naval Postgraduate School Monterey, CA 93943-5000	1
6.	Dan Sakoda, (Code SP) Space Systems Academic Group Naval Postgraduate School Monterey, CA 93943-5000	1
7.	Jim Horning, (Code SP) Space Systems Academic Group Naval Postgraduate School Monterey, CA 93943-5000	1
8.	Tony Ranaweera, (Code SP) Space Systems Academic Group Naval Postgraduate School-5000 Monterey, CA 93943	1